МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

В.Я. Юрчишин

Технології суперкомп'ютерних обчислень

ЛЕКЦІЇ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для студентів, які навчаються за спеціальністю 121 «Інженерія програмного забезпечення» (освітня програма «*Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем*»)

Київ
«КПІ ім. Ігоря Сікорського»
2022

Рецензенти: *Зубчукчук Віктор Іванович, канд. техн. наук, доц.*
Тимошин Юрій Афанасійович, канд. техн. наук, доц.
Відповідальний
редактор *Заболотня Т.М., канд. техн. наук, доц.*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол №1 від 02.09.2022 р.) за поданням Вченої ради факультету прикладної математики (протокол № 1 від 01.09.2022 р.)

Електронне мережне навчальне видання

Юрчишин Василь Якович, к. т. н, доц.

Технології суперкомп'ютерних обчислень

ЛЕКЦІЇ

Технології суперкомп'ютерних обчислень: лекції [Електронний ресурс]: навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «*Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем*»)/ В.Я.Юрчишин; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 7,32 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022. – 254 с.

Навчальний посібник розроблено для вивчення студентами теоретичних відомостей та ознайомлення з практичними прийомами роботи по розробці програмних продуктів в області суперкомп'ютерних обчислень.

Навчальне видання призначене для студентів, які навчаються за спеціальністю 121 Інженерія програмного забезпечення (освітня програма «*Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем*») освітньо-кваліфікаційного рівня Магістр факультету прикладної математики НТУУ КПІ ім. Ігоря Сікорського.

ЗМІСТ

ВСТУП

Застосування інформаційних технологій на базі суперкомп'ютерних систем сприяють розробці сучасного програмного забезпечення, прикладом якого можуть виступати широкомасштабні системи моніторингу, керування та аналізу з глобально розподіленими джерелами даних. Значення суперкомп'ютерних технологій є одними з найважливішими в світі, завдяки їм границі країн, відстані та фізичні обмеження перестають мати минулі значення. Суперкомп'ютери по праву вважаються найпотужнішою інформаційною зброєю сьогодення.

У даному навчальному посібнику розглядаються теоретичні та практичні основи технологій суперкомп'ютерних обчислень з використання технологій паралельних та розподілених обчислень, віртуалізації серверних систем, проектування корпоративних обчислювальних систем та застосування кластерних та гетерогенних розподілених обчислювальних систем із використанням мережевих протоколів взаємодії клієнтських та серверних додатків. управління та аналізу з глобально розподіленими джерелами даних.

Навчальна дисципліна належить до складу вибіркових дисциплін підготовки студентів спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем») освітньо-кваліфікаційного рівня Магістр.

Метою конспекту лекцій є отримання необхідного рівня знань і придбання практичних умінь і навичок в використанні інформаційних технологій на базі суперкомп'ютерних систем.

Курс лекцій розрахований на 36 академічних годин аудиторних занять. Посібник складається з 18 лекцій. В кожній лекції приводяться теоретичні відомості з певної теми, контрольні питання для самоконтролю та список рекомендованої літератури.

1 Суперкомп’ютер. Етапи розвитку суперкомп’ютерних технологій

1.1 Вступ до дисципліни. Що таке суперкомп’ютер?

Суперкомп'ютером називається обчислювальна система, обчислювальна швидкодія якої багаторазово вища, ніж у сучасних їй комп'ютерів масового випуску, це машина, орієнтована на обчислення, на значне, хоча б на порядок зниження часу виконання складних розрахунків у порівнянні зі звичайними комп'ютерами.

Тому суперкомп'ютер - виріб дрібносерійний, або навіть штучний і при його виробництві застосовуються технології і підходи, які мають високу вартість. Поняття суперкомп'ютера з'явилося, фактично, одночасно з початком серійного випуску комп'терів. Якими ж факторами визначається конкретний вибір тих чи інших технологічних підходів до створення суперкомп'ютеру?

Тисячі розробників суперкомп'ютерів у всьому світі намагаються створювати обчислювальні системи набагато швидші, ніж комп'ютери серійного випуску. Тим часом над вдосконаленням комп'ютерів серійного випуску працюють сотні тисяч, якщо не мільйони, людей. Як першим обігнати других? Розробникам суперкомп'ютерів необхідно працювати дуже швидко. Це можливо лише в разі, якщо вони будуть максимально запозичувати рішення, технології і компоненти з комп'ютерної індустрії серійного випуску, з тим, щоб скомбінувати їх по-новому, точково вкласти власні сили в виготовлення лише відсутніх фрагментів системи і отримати бажану нову якість - суперкомп'ютер .

Але технічно виправдані способи запозичення змінюються. Разом з ними дуже швидко змінюється і вигляд суперкомп’ютерів, тобто відбувається чергова суперкомп'ютерна революція.

Правило економії сил на розробку суперкомп'ютера був істотним завжди. З переходом (у 80-х роках XX ст.) комп'ютера в категорію масових виробів, які випускаються багатомільйонними тиражами, цей фактор став вирішальним. Про це необхідно постійно пам'ятати, аналізуючи і порівнюючи витончені конструкції, які претендують на звання Суперкомп'ютерних архітектур. Зовнішність суперкомп'ютера на кожному етапі технологічного розвитку визначається Правилом економії сил розробника.

Приблизно до кінця 80-х років XX ст. вдавалося будувати суперкомп'ютери в рамках фоннеймановської архітектури. Ускладнюючи внутрішній устрій процесора, можна було змусити його виконувати приблизно (або навіть в точності) такі ж послідовності машинних команд, що і «звичайний» процесор, у багато разів швидше. Іншими словами, фоннеймановська архітектура залишалася адекватною формою організації зростаючої кількості електронних елементів, придатних для інтеграції в єдину обчислювальну систему.

На цьому етапі:

- програміст майже не замислювався над тим, для суперкомп'ютера чи для «звичайного» комп'ютера він пише програму;
- розробник обладнання суперкомп'ютера розробляв «суперпроцесор»;
- суперкомп'ютер таким чином представляв собою звичайну фоннеймановськау машину, оснащену «суперпроцесором».

Сама можливість значно прискорювати процесор в порівнянні з серійними зразками, удосконалюючи лише його внутрішній устрій, забезпечувалася зростанням кількості електронних елементів, які використовуються при виготовленні «звичайного» процесора.

З появою мікропроцесорів епоха «суперпроцесорів» закінчилася і єдиним способом виготовлення суперкомп'ютера стало виготовлення паралельної обчислювальної системи, тобто машини, в якій багато процесорів спільно трудяться над одним завданням. Такі машини стали будувати з серійних мікропроцесорів, але на оригінальній схемотехнічній базі (шафи, плати, лінії зв'язку). Цей технологічний переворот вважають першою суперкомп'ютерної революцією.

Можливість отримувати суперкомп'ютер за допомогою розробки власних засобів інтеграції стандартних мікропроцесорів в єдину систему існувала до тих пір, поки серійні засоби інтеграції, тобто обладнання локальних мереж, були розвинені порівняно слабко і для інтеграції процесорів «на суперкомп’ютерному рівні» не годилися. В міру вдосконалення обладнання локальних мереж і інтерфейсів і їх підключення до серійних комп'ютерів і ця ситуація змінилася. Приблизно в кінці 90-х років серійне мережеве обладнання дозволило використовувати в якості суперкомп'ютера звичайну локальну мережу, компактно розташовану і оснащену відповідним програмним забезпеченням. Технічний і економічний сенс розробки і виготовлення власних плат і шаф для інтеграції стандартних мікропроцесорів в єдину систему пропав. Переважна більшість суперкомп'ютерів стало будуватися за технологією кластерів виділених робочих станцій, отже збиратися методами системної інтеграції з готових серійних комп'ютерів загального призначення, плат і комутаторів локальних мереж [2]. Цей переворот називають другою суперкомп'ютерної революцією.

На цьому етапі:

- для програміста, в порівнянні з попереднім етапом, нічого принципово не змінилося, крім того, що таких програмістів стало багаторазово більше, як і самих суперкомп'ютерів;
- розробник суперкомп'ютерного обладнання змушений був стати системним інтегратором - розробляти стало нічого, всі використовувані компоненти і засоби їх інтеграції стали випускатися великими серіями;

- суперкомп'ютер, в результаті, з паралельної обчислювальної системи на оригінальній схемотехнической базі став кластером виділених робочих станцій [2, 3].

Сама по собі ідея використовувати локальну мережу в якості суперкомп'ютера значно старше другої суперкомп'ютерної революції. У проміжку між першою і другою суперкомп'ютерними революціями локальні мережі бурхливо розвивалися, в тому рахунку в організаціях, зацікавлених в високопродуктивних обчисленнях. Природньо, робилися спроби використовувати локальні мережі загального призначення, наприклад, під час нічних простоїв, для паралельних обчислень. Такі локальні мережі за своїми комунікаційними можливостями значно поступалися «справжнім» суперкомп'ютерам, але на деяких, які не потребують особливо інтенсивних комунікацій між вузлами, паралельні програми вдавалося успішно виконувати. Саме тоді з'явилася сама назва «обчислювальний кластер», і перші варіанти програмного забезпечення, які дозволяли використовувати локальну мережу в цій якості. Це програмне забезпечення було орієнтоване на паралельне використання локальної мережі в обох режимах: в основному - як локальної мережі загального призначення, і лише іноді - як обчислювального кластеру. Отже, обладнання не виділялося спеціально для використання в режимі обчислювального кластеру. Друга суперкомп'ютерна революція, як ми тільки що бачили, полягала в тому, що при необхідності виділити обладнання для використання виключно в якості обчислювального кластера, це обладнання стало оформлятися також у вигляді локальної мережі. На відміну від кластерів попереднього етапу, такі кластери стали забезпечуватися абсолютно особливим програмним забезпеченням, часто виключаючим використання вузлів як машин загального призначення (на вузлі сучасного обчислювального кластеру електронну почту не відправляють, і наукові статті готують).

Щоб розрізняти два способи організації паралельних обчислень на обладнанні локальних мереж і, відповідно, два принципово різних способи організації програмного забезпечення, «нові» кластери стали називати кластерами виділених робочих станцій, а старі, відповідно, кластерами невиділених робочих станцій. Програмні технології організації кластерів невиділених робочих станцій існують і сьогодні.

Необхідність економії сил проходить «червоною ниткою» через всю ієрархію суперкомп'ютерних технологій, зачіпаючи насамперед моделі та парадигми паралельного програмування і чисельні методи паралельного вирішення завдань. Порівнюючи моделі і технології паралельного програмування, це проглядається особливо ясно.

Приблизно до середини 80-х можна було з високим ступенем точності стверджувати, що суперкомп'ютер - це комп'ютер, оснащений спеціальним,

особливо потужним (і тому - особливо дорогим) процесором. Ситуація радикально змінилася буквально за 2-3 роки, коли прогрес мікроелектроніки дозволив подолати рубіж мільйона транзисторів на одному кристалі. Стало ясно, що технологіям виготовлення процесорів з окремих елементів прийшов кінець - відтепер всі процесори будуть тільки однокристальними мікропроцесорами. У 1989 році фірма Intel випустила однокристальний мікропроцесор, який зрівнявся за продуктивністю з чинним тоді ж суперкомп'ютером Сгау-1 [2]. Оскільки виробництво мікросхем, за своєю природою, є конвеєрним і досить дешевим, подолання цього рубежу означало кінець ери особливо потужних процесорів.

В цих умовах для підвищення швидкодії понад «масово доступного» рівня, тобто для створення суперкомп'ютера, залишився єдиний шлях. - шлях об'єднання багатьох процесорів для паралельного розв'язання одного особливо складного завдання. З цього часу і до наших днів ми можемо впевнено ставити знак рівності між суперкомп'ютером і паралельною (або багатопроцессорною) обчислювальною системою [1]. Справедлівості заради слід зазначити, що звичайний однопроцесорній комп'ютер теж є паралельних обчислювальних системою, бо ще сам фон Нейман наполягав на тому, що операції над числами в його машині мають виконувати паралельно на рівні двійковіх розрядів- не розряд за розряд (послідовно), а всі розряди зразу (паралельно). Ми настілько звіклі до фоннеймановського комп'ютера, що перестали його вважати паралельною системою. І це вірно, бо все відносно.

Комп'ютери розділилися на машини масового (за поняттями 80-х) виробництва і машини рекордної продуктивності, які випускаються окремими екземплярами або малими серіями. Останні стали називати суперкомп'ютерами, а способи їх побудови і використання - технологіями високопродуктивних обчислень. До деяких пір ці технології розвивалися в рамках фоннеймановская принципів, але в кінці 80-х, коли це перестало бути можливим, суперкомп'ютери стали паралельними обчислювальними системами, а технології високопродуктивних обчислень стали невіддільними від технологій паралельної обробки даних.

Можна відзначити, як мінімум, два цікавих моменти для результату «першої суперкомп'ютерної революції».

• по-перше, ключовою технологією в розробці суперкомп'ютерів стала (замість технології розробки процесора) технологія комплексування багатьох процесорів в єдину систему;

• по-друге, суперкомп'ютер став архітектурно відрізнятися від традиційних машин масового випуску набагато сильніше, ніж раніше, а це вже не могло не відбитися на технологіях застосування. Для того, щоб паралельна машина дала

швидкісний виграш, треба написати паралельну програму, а це зовсім не те ж саме, що написати програму для звичайного комп'ютера.

До середини 90-х розвиток багатопроцесорних суперкомп'ютерів йшов по двох паралельних напрямах. Магістральний напрям був пов'язаний з побудовою спеціалізованих багатопроцесорних систем з мікросхем масового випуску. Головним питанням при побудові таких систем було питання про спосіб комплексування процесорів.

Найбільш зручним для програміста був би спосіб об'єднання багатьох процесорів на єдиній, симетрично адресуємій загальній пам'яті. Такі комп'ютери отримали назву SMP-систем (SMP - Symmetric Multi Processing). Головне достоїнство SMP-систем з точки зору програміста полягає в тому, що для організації паралельної роботи потрібно поділити між процесорами лише обчислення, але не дані: дані і так належать «всім на рівних». На жаль, цей самий зручний для програміста спосіб є, одночасно, і найдорожчим в апаратній реалізації. Процесор спілкується з власною пам'яттю вельми інтенсивно і для того, щоб пам'ять могла обслуговувати потреби багатьох процесорів, вона повинна бути або дуже складною і дорогою, або буде досить повільної з точки зору кожного з процесорів. Коли число процесорів виростає до десятків, завдання взагалі дуже ускладнюється. Таким чином, область застосування SMP-підходу - дорогі, не дуже великі за кількістю вузлів системи, але досить зручні для програмування.

Менш зручним для програміста, але на порядок дешевшим і допускючим потенційно нескінченне масштабування, виявився підхід на основі об'єднання окремих самостійних комп'ютерів спеціалізованими каналами зв'язку. Говорячи про окремі комп'ютерах, маємо на увазі не конструкцію, а лише логіку побудови системи. Насправді використовувалися не комп'ютери загального призначення, з окремим екраном, клавіатурою і блоком живлення, а спеціалізовані обчислювальні модулі, компактні, оснащені спеціалізованою зв'язковою апаратурою, яка не застосовується в системах загального призначення. Модулі ці розроблялися спеціально для використання в якості елементів конкретного суперкомп'ютера і, найчастіше, ні в якому іншому використані бути не могли. При цьому логічно кожен такий модуль являв із себе повноцінний комп'ютер, тобто мав у своєму складі окремий процесор і виконував абсолютно окрему, незалежну від інших програму, яка оперує своїми локальними даними. Для організації взаємодії модулів між собою в кожній такій програмі слід було явно закодувати акти передачі даних по каналах зв'язку.

Такі системи, на відміну від SMP, отримали назву MPP (MPP - Massively Parallel Processing, назва підкреслює теоретично необмежену масштабованість систем цього класу) [1, 3]. Найбільш логічно завершеного вигляду концепція

систем MPP досягла в трансп’ютерних системах, які не отримали згодом подальшого розвитку [2].

Проміжне становище між системами SMP і MPP займають системи NUMA (NUMA - Non-uniform Memory Access). Це системи, в яких загальна для багатьох процесорів пам'ять присутня (апаратно або логічно), але за своїми архітектурними властивостям (швидкодії, перш за все) настільки сильно відрізняється від локальної пам'яті процесора, що це необхідно явно враховувати при програмуванні.

І за ціною, і за стилем програмування ці системи ближче до SMP, ніж до MPP [1].

Логічно MPP-система мало відрізняється від звичайної локальної мережі, а локальні мережі в описуваний період переживали досить бурхливий розвиток. Виникало цілком природне бажання спробувати використовувати локальну мережу як паралельний суперкомп’ютер - наприклад, в періоди нічних простоїв. Вся побудова суперкомп'ютера при цій технології зводиться до розробки відповідного програмного забезпечення.

Такі локальні мережі, які використовуються в якості багатопроцесорних обчислювальних систем, стали називати кластерами робочих станцій або просто «кластерами». Колись рівень розвитку апаратури локальних мереж був досить низький. З цієї причини кластери сильно програвали справжнім, традиційним суперкомп'ютерам в своїх комунікаційних можливостях. На практиці це означало, що клас успішно вирішуємих на кластерах завдань був набагато вужчим, ніж на «справжніх» MPP. Справді, чим тісніше здатні взаємодіяти вузли багатопроцесорної системи, тим більше шансів на успіх має програміст, який намагається налагодити одночасну, спільну роботу декількох процесорів над одним завданням.

У другій половині 90-х грянула «друга суперкомп'ютерна революція», пов'язана з двома досягненнями в області апаратури великосерійного випуску. Повсюдне оснащення персональних комп'ютерів високошвидкісною шиною PCI [12] і перехід на дешеву, але досить швидку (комутуємую) мережу Fast Ethernet, привели до того, що кластери «наздогнали» спеціалізовані MPP-системи за своїми комунікаційним можливостям. При цьому серед виробників спеціалізованого комунікаційного оболаднання, яке раніше застосовувалося тільки в складі MPP- систем стало прийнято оформляти свої вироби у вигляді стандартних плат PCI, які встановлюються в комп'ютери загального призначення, а не у вигляді наборів мікросхем, з яких потрібно виготовляти спеціалізовані модулі.

У цих умовах виготовлення MPP-систем зі спеціалізованих, тільки для цієї мети спроектованих і виготовлених модулів втратило будь-який сенс. За рахунок деяких втрат в щільності компонування стало можливим виготовити

MPP-систему не зі спеціальних модулів, а зі стандартних персональних комп'ютерів, куплених в магазині і при необхідності оснастити ці модулі дуже високопродуктивним, спеціалізованим комунікаційним середовищем, так як відповідні пристрої теж можна було придбати . При цьому можна було бути абсолютно впевненим, що, якщо тільки такий виріб існує, за форматом він є звичайною PCI-платою, яку можна вставити в стандартний персональний комп'ютер.

Різниця між MPP та кластером, таким чином, з області апаратної змістилася в область чисто організаційну [2].

Локальні мережі, спеціально зібрані для використання в якості багатопроцесорної обчислювальної системи, компактно розміщені в одному або декількох шафах, стали називати кластерами виділених (dedicated) робочих станцій, підкреслюючи цією назвою, що вузли такої мережі виділені для роботи в складі суперкомп'ютера і не використовуються для розрахунків зарплати, ділового листування, або комп'ютерних ігор.

У той же час, кластери в старому сенсі цього слова не припинили свого існування, тільки тепер їх стали називати кластерами невиділених робочих станцій.

На відміну від традиційних (невиділених) кластерів, кластери виділених робочих станцій стали оснащувати програмним забезпеченням, орієнтованим виключно на управління установкою як єдиним цілим: так, на грамотно організованому виділеному кластері окремий вузол зазвичай не має адреси в Інтернеті і не може використовуватися як самостійний комп'ютер загального призначення. Це дозволяє нам говорити про кластери виділених робочих станцій як про повноцінні MPP- системи, які відрізняються від MPP-систем попереднього покоління лише апаратною архітектурою, але не порядком використання.

У кластерах невиділених робочих станцій, навпаки, програмне забезпечення орієнтоване на використання набору абсолютно самостійних машин, при необхідності, не тільки за прямим призначенням, але ще і в якості обчислювальних вузлів паралельної обчислювальної системи.

На розвиток систем SMP і NUMA друга суперкомп'ютерна революція практично не відбилася, ці системи продовжували свій розвиток в класі дуже дорогих і, як правило, не дуже великих машин. MPP- системи зі спеціалізованих модулів практично зникли в нових розрбках, поступившись місцем кластерам виділених робочих станцій. Втім, багато раніше побудовані MPP-системи. тим не менше, продовжують експлуатуватися [1, 3].

Нарешті, ціновий розрив між системами на базі спеціалізованої апаратури, з одного боку, і кластерами виділених робочих станцій, з іншого, становить, при порівняній кількості процесорів, приблизно два порядки і продовжує зростати.

Цим обумовлено повсюдне поширення кластерів виділених робочих станцій в якості технології побудови порівняно дешевих і доступних суперкомп'ютерів.

Сформована в ході «другої суперкомп'ютерної революції» структура світового парку суперкомп'ютерів, судячи з усього, збережеться в найближчому майбутньому.

Таким чином, слова «кластер», «паралельна система» і «суперкомп'ютер» в значній мірі є синонімами, але не «за визначенням», а в силу того циклу технологічної еволюції апаратури, який ми коротко розглянули. Слід зазначити, що словом «кластер» прийнято позначати, крім паралельної обчислювальної системи, взагалі будь-яке утворення з декількох комп'ютерів, об'єднаних для вирішення єдиного завдання. Наприклад, якщо сервер системи резервування авіаквитків реалізований на базі локальної мережі, з метою підвищення продуктивності та відмовостійкості, то це - кластер. Далі ми будемо розуміти під словом «кластер» саме паралельну обчислювальну систему, побудовану за кластерної технології, а не всякий кластер взагалі.

Розглянуті дві суперкомп'ютерні революції дають нам два приклади конкретних рушійних сил, що породжують принципово нові технічні рішення. Найбільш масштабні зміни відбуваються з причини зростання щільності напівпровідникових структур на кристалі мікросхеми (перша суперкомп'ютерна революція). Такі ж масштабні зміни відбуваються, коли в світі комп'ютерів масового випуску різко змінюються комунікаційні, мережеві технології (друга суперкомп'ютерна революція). Це, однак, зовсім не скасовує того факту, що на протязі всього «кластерного» періоду такими фірмами, як Cray і Silicon, інтенсивно і успішно розвивалися технології побудови суперкомп'ютерів на оригінальній схемотехнічній (і навіть елементній) базі. При цьому ступінь кількісного переважування кластерної технології над технологіями Cray і Silicon в повній мірі з'ясувалася «заднім числом», на досвіді вже пройшовшого «кластерного» періоду. Приблизно так само слід ставитися до пропонованих міркувань про третю суперкомп'ютерну революцю. Зазначимо два взаємопов'язаних, але все ж досить автономних напрямках третьої суперкомп'ютерної революції. Перший напрямок - докорінна зміна комунікаційних технологій. Другий напрямок полягає у змінах в організації самих обчислювачів, бо сьогоднішні технологічні норми мікроелектроніки досягнули кількох одиниць нанометрів, що свідчить про припинення прогресу в напрямку подальшої мікромініатюризації елементної бази.

На чому хотілося б особливо загострити увагу - це досить суттєва трудомісткість освоєння суперкомп'ютера в порівнянні з традиційним комп'ютером.

Методів автоматичного перетворення програм для традиційних комп'ютерів в програми для паралельних машин, що дають при виконанні

швидкісний виграш, не просто не існує, а не може існувати взагалі, в крайньому разі , в найближчому майбутньому. Отже, користувач, який зважився на освоєння суперкомп'ютера, має бути морально готовий до того, що доведеться вчитися, витрачати час і сили. Це стосується навіть тих користувачів, які не пишуть програм самі, а займаються виключно експлуатацією готових прикладних програм, придбаних «під ключ» - хоча, звичайно, в помітно меншій мірі, ніж просто програмістам.

Освоєння цього курсу - це для тих, кому треба багато і швидко рахувати, причому «треба» настільки сильно, що вони готові вкласти в прискорення рахунку помітний обсяг робочого часу і власних сил. Під масовими обчисленнями ми розуміємо, в першу чергу, інженерно-технічні розрахунки - словом, чисельне рішення задач математичної фізики у всіх видах і варіантах. Крім того, суперкомп'ютери застосовуються (і досить широко) в обчислювально трудомістких завданнях обробки даних нечислового характеру, наприклад, в додатках з області data mining, але це - окрема тема, якої ми торкатися не будемо.

Далі доречно зупинитися на поширеному омані. Йдеться про думку, що з настанням «кластерного» етапу в суперкомп'ютерних технологіях, для виготовлення суперкомп'ютера досить взяти шафу (стійку), яка вміщує необхідну кількість системних блоків («лез»). На жаль, це не просто не зовсім вірно - це зовсім не вірно. Суперкомп'ютер не є проста сума складових його готових компонентів.

Викладені в цій лекції відомості по природі своїй дуже динамічні. Не тільки видані традиційним способом книги, а й мережеві джерела знань в даній області стрімко старіють. Навіть заміна традиційних посилань на «паперові» видання посиланнями на той чи інший мережевий ресурс не вирішує проблеми - мережеві ресурси можуть з'являтися і зникати. Передбачається, що студент буде використовувати автоматичні індексуючі мережі пошукові машини, наприклад, Google, а не сайти з занесенням посилань.

Литеретура

1. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. Учебное пособие – М.: Издательство Московского университета, 2011. – 563с .
2. Лацис А. О. Параллельная обработка данных : учеб. пособие для студ.вузов / — М.: Издательский центр «Академия», 2012. — 336 с. —
3. Воеводин В. В. Параллельные вычисления. — СПб. : БХВ- Петербург, 2002. — 608 с.
4. http7/www. top500.org.

5. http7/www. vmware. com.
6. http7/www. intel. com.
7. What is grid computing? - Gridcafe. E-sciencecity.org. Retrieved 2013-09-18.

1.2 Тенденції розвитку та характеристики суперкомп'ютера. Top 500.

У сучасному світі суперкомп'ютери стали робочим інструментом в наукових дослідженнях та інженерних розрахунках. Їх по праву назвають основною технологічною зброєю XXI ст.

Необхідність у високопродуктивних обчислювальних системах зі значним обсягом пам'яті обумовлена ускладненням математичних моделей за рахунок врахування великої кількості досліджуваних факторів, використанням системного підходу в моделюванні (моделюється не один об'єкт, а одночасно група об'єктів та явищ, або середовище в цілому), реалізацією динамічної візуалізації результатів розв'язування прикладних задач та інше. Суперкомп'ютери відіграють важливу роль при розв'язуванні економічних, екологічних, біоінженерних, медичних, бюджетно-фінасових, науково-технічних, оборонних загальнодержавних та інших задач. Вони використовуються для побудови і дослідження кліматичних моделей, прогнозування змін в атмосфері та гідросфері Землі, моделювання побудови білка з амінокислот, аналізу результатів діагностики захворювань людини, розробки нових ліків, проектування нових матеріалів, у генетичному моделюванні тощо, тобто там де для успішного вирішення вагомих прикладних задач і досягнення якісно нових результатів потрібні комп'ютери з надзвичайно високою продуктивністю.

В сучасному світі використовуються різні типи суперкомп'ютерів. На початку 1990 р в иникла необхідність у порівнянні характеристик суперкомп'ютерів, щоб можна було відслідковувати тенденції розвитку високопродуктивних обчислювальних систем. Такий проект був запущений у 1993 р. Цей проект отримав назву Top500 [1].

Проєкт, заснований з метою відслідковування та виявлення трендів у сфері HPC (High Performance Computing). High Performance Computing — комп'ютерні технології з надзвичайно високою продуктивністю (порівняно зі звичайними ПК масового вжитку), які використовують для вирішення масштабних проблем у науці, інженерії та бізнесі [2]. Двічі на рік на веб-сайті Top500 публікують список п'ятисот найбільш потужних комп'ютерних систем. Рейтинг складають на основі випробування систем за методикою Linpack.

Для кожної комп'ютерної системи виділяють наступні числові характеристики (отримані за допомогою Linpack):

- Rmax: швидкодія в GFLOPS для найбільшої задачі, розв'язаної машиною;

- Nmax: розмір найбільшої задачі, розв’язаної машиною.
- N1/2: розмір задачі, за якого досягнуто половини швидкості, яка відповідає Rmax.
- Rpeak: теоретична гранична (пікова) швидкодія (theoretical peak performance) в GFLOPS для машини.

Ці результати і використовують для формування рейтингу Top500.

Top500 – проект для визначення рейтингу і опису 500 найпотужніших суспільно відомих комп’ютерних систем світу. З червня 1993 р. він складається два рази на рік (у червні та листопаді). Ознайомитися з ним можна на сайті http://www.top500.org.

Порівняння суперкомп’ютерів ведеться за наступними параметрами.

По-перше, існуючі суперкомп’ютери мають кластерну архітектуру. Кластер – це масив вузлів , які з’єднуються між собою в деяку особливу комунікаційну мережу. Для кожного кластера існує виділений керуючий вузол. Обчислювальні вузли мають свою оперативну пам’ять і працюють під керуванням своєї операційної системи. Вузли збираються з великої кількості мікропроцесорних ядер. Окремий мікропроцесор може мати одне, два, чотири і більше мікропроцесорних ядер, тому в рейтингу фігурує загальна кількість ядер. В кластер можуть входити тисячі, сотні тисяч мікропроцесорних ядер, однак із зростанням кількості ядер зростають труднощі в програмуванні. Суперкомп’ютери складаються з тих самих ядер, що й інші комп’ютери. Кількість мікропроцесорів у суперкомп’ютерах нині складає 0,1% від загальної кількості мікропроцесорів, які використовуються в комп’ютерах.

По-друге, розрізняють потенційну або пікову Rpeak швидкодію суперкомп’ютера і реальну Rmax, які вимірюються за результатами стандартного тесту Linpack. Linpack – це сукупність підпрограм для розв’язування систем лінійних алгебраїчних рівнянь великої розмірності.

Швидкодія суперкомп’ютера вимірюється в терафлопсах (10^{12} флопс), а з 2008 р. в петафлопсах (10^{15} флопс). Флопс – це величина для вимірювання продуктивності комп’ютера і показує кількість операцій з плаваючою комою за секунду, яку виконує обчислювальна система.

Реальна швидкодія залежить від багатьох факторів, зокрема, від якості системного і прикладного забезпечення, пропускної здатності між’ядерних з’єднань, температурного режиму і т.д.

По-третє, відношення реальної швидкодії до потенціально можливої характеризує коефіцієнт корисної дії (ККД) суперкомп’ютера. Чим вищий ККД, тим більш ефективна робота суперкомп’ютера.

Отже, кількість мікропроцесорних ядер, рівень реальної і потенціальної швидкодії, а також ККД складають найважливіші характеристики суперЕОМ.

Проаналізувати суперкомп'ютери у світовому рейтингу за весь період їх існування неможливо. Спробуємо зробити зріз середини цього періоду і наведемо досягнення сьогодення.

За рейтингом, складеним в листопаді 2009 р., першими десять суперкомп'ютерів та їхні характеристики є наступні (таблиця 1).

Таблиця 1. Перші 10 позицій у рейтингу суперкомп'ютерів (34 редакція)

Місце	Організація, країна	Комп'ютер/Рік випуску Виробник	К-ть ядер	Продуктивність	
				R_{max} TF	R_{peak} TF
1	Oak Ridge National Laboratory США	Jaguar - Cray XT5-HE Opteron Six Core 2,6 GHz / 2009 Cray Inc.	224162	1759,00	2331,00
2	DOE/NNSA/LANL США	Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3,2 Ghz / Opteron DC 1,8 GHz, Voltaire Infiniband / 2009 IBM	122400	1042,00	1375,78
3	National Institute for Computational Sciences/University of Tennessee США	Kraken XT5 - Cray XT5-HE Opteron Six Core 2,6 GHz / 2009 Cray Inc.	98928	831,70	1028,85
4	Forschungszentrum Juelich (FZJ) Німеччина	JUGENE - Blue Gene/P Solution / 2009 IBM	294912	825,50	1002,70
5	National SuperComputer Center in Tianjin/NUDT Китай	Tianhe-1 - NUDT TH-1 Cluster, Xeon E5540/E5450, ATI Radeon HD 4870 2, Infiniband / 2009 NUDT	71680	563,10	1206,19
6	NASA/Ames Research Center/NAS Китай	Pleiades - SGI Altix ICE 8200EX, Xeon QC 3,0 GHz/Nehalem EP 2,93 GHz / 2009 SGI	56320	544,30	673,26
7	DOE/NNSA/LLNL США	BlueGene/L - eServer Blue Gene Solution / 2007 IBM	212992	478,20	596,38
8	Argonne National Laboratory США	Blue Gene/P Solution / 2007 IBM	163840	458,61	557,06
9	Texas Advanced Computing Center/Univ. of Texas США	Ranger - SunBlade x6420, Opteron QC 2,3 GHz, Infiniband / 2008 Sun Microsystems	62976	433,20	579,38
10	Sandia National Laboratories / National Renewable Energy Laboratory США	Red Sky - Sun Blade x6275, Xeon X55xx 2,93 GHz, Infiniband / 2009 Sun Microsystems	41616	423,90	487,74

Лідером рейтингу серед 500 найпотужніших суперкомп'ютерів світу на листопад 2009 р. був суперкомп'ютер Jaguar XT5 (раніше цей рейтинг очолювала система IBM Roadrunner). Він знаходиться в США в національному центрі обчислювальних наук в Окріджі (штат Теннесі) і

використовується, зокрема, для обчислення руху космічних тіл, побудови кліматичних моделей, дослідження атомів і т.д.

Суперкомп'ютер Jaguar XT5 побудований на базі платформи Cray. Jaguar складається з 37376 шестиядерних процесорів AMD Opteron, так що загальна кількість ядер складає 224256, обсяг оперативної пам'яті – 299 Тб, об'єм дискової пам'яті – 10 тисяч Тб і займає площу 392 м2.

Згідно з методологією тестування Linpack, Jaguar працює з реальною швидкістю 1,759 петафлопс. Це приблизно в 29464 рази більше, ніж найпродуктивніший суперкомп'ютер у першій редакції рейтингу Top500 (червень 1993 р.) Пікова продуктивність системи Cray складає 2,3 петафлопс (див. табл.1).

Друге місце в цьому рейтингу займав суперкомп'ютер Roadrunner виробництва корпорації IBM. Він складається з 6500 двоядерних процесорів і має потужність 1,105 петафлопс. Слід зазначити, що першим суперкомп'ютером, який на тесті Linpack показав продуктивність на рівні петафлопс був суперкомп'ютер IBM Roadrunner (з'явився у червні 2008 р.).

На третьому місці знаходився суперкомп'ютер Kraken, який, як і Jaguar, побудований на базі Cray XT5. Його продуктивність 832 терафлопс.

На четвертій позиції – суперкомп'ютер IBM Blue Gene/P з обчислювальною потужністю 825,5 терафлопс, на п'ятому – Tiange-1 (найпотужніший суперкомп'ютер в Китаї). Tiange-1 створений на основі 6144 процесорів Intel і 5120 графічних процесорів AMD з продуктивністю на тесті Linpack 563,1 терафлопс. Як видно з таблиці 1, вісім комп'ютерів першої десятки знаходяться в США, два інших – в Китаї та Німеччині.

Останнє місце в цьому рейтингу Top500 належить суперкомп'ютеру Blade Center JS21 (Великобританія) з характеристиками 2880 ядер, Rmax = 20,05 терафлопс, Rpeak= 28,00 терафлопс, що за показником Rmax в 88 разів менше, ніж в суперкомп'ютера-лідера Jaguar XT5. І взагалі, в усіх Top500 спостерігається приблизно 100-кратний діапазон для Rmax першого і останнього суперкомп'ютера рейтингу.

Основним лідером в списку рейтингу Top500 за всі роки в цей період, починаючи з 1993 р є США, п'ять разів в лідери виходила Японія (1996, 2002, 2003, 2004, 2005).

Щорічні листопадові дані з продуктивності суперкомп'ютерів -лідерів за вказаний період, починаючи з 1993 р., наведені в таблиці 2.

Таблиця 2

Рік	1993	1994	1995	1996	1997	1998	1999	2000
R_{max} TF	0,124	0,170	0,170	0,368	1,338	1,338	2,379	4,938
R_{peak}TF	0,226	0,236	0,236	0,614	1,830	1,830	3,207	12,228
2001	2002	2003	2004	2005	2006	2007	2008	2009
7,226	35,86	35,86	70,72	280,6	280,6	478,2	1105,0	1759,0
12,288	40,96	40,96	91,75	367,0	367,0	596,38	1456,7	2331,0

За цей період (1993-2009р) розвиток суперкомп'ютерів продемонстрував різке зростання їх обчислювальної потужності: навіть звичайний суперкомп'ютер, який у 2009 р. зайняв у рейтингу останнє місце, у 18 разів перевищив сумарну потужність усіх 500 суперЕОМ в 1993 р.
Якщо брати сукупну потужність 500 тодішних суперкомп'ютерів світу, то вона досягла майже 27 петафлопс, що в 24916 разів більше аналогічного показника 1993р.

Але, не дивлячись на те, що перше місце належало суперкомп'ютеру фірми Cray, більшість з 500 тодішних найпродуктивніших суперкомп'ютерів, як і раніше, були зібрані двома компаніями: 209 (41,8%) фірмою HP, 185 (37%) – IBM. Далі в рейтингу виробників знаходилися фірми Cray та SCI, які випусили по 19 (3,8%), Dell – 16 (3,2%), Sun Microsystems – 11 (2,2%), Appro international – 6 (1,2%), Fujitsu та Bull SA – по 5 (1,0%), Hitachi – 3 (0,6%), NEC – 2 (0,4%) суперкомп'ютери. У 80,4% випадків для суперкомп'ютерних систем використовувалися процесори фірми Intel, у 8,4% – фірми AMD. Згідно з поточним рейтингом 90% суперкомп'ютерів використовують операційні системи на базі ядра Linux.

Світовий рейтинг країн за кількістю суперкомп'ютерів наведений в таблиці 3, хоча більш суттєвим було б порівняння рейтингу країн за Rmax, який підтверджується реальним тестовим випробуванням. В зазначену 34 редакцію рейтингу входили 32 країни. Безперечним лідером за цими двома показниками були США. Доля США в загальній кількості суперкомп'ютерів світового рейтингу становить 55,4%. Друге та третє місця ділили між собою Великобританія та Німеччина: Великобританія займала другу позицію за кількістю суперкомп'ютерів, однак Німеччина займала друге місце в рейтингу сумарної реальної продуктивності своїх суперкомп'ютерів. Аналогічна ситуація з четвертим і п'ятим місцями, їх поділяли між собою Франція і Китай. Франція випереджала Китай за кількістю суперкомп'ютерів,
проте Китай випереджав Францію за сумарною продуктивністю суперкомп'ютерів.

Таблиця 3 Рейтинг країн на 2009р за кількісттю суперкомп'ютерів

Країни	К-ть	Доля %	Продуктивність		К-ть процесорів
			R_{max} (GF)	R_{peak} (GF)	
США	277	55,40%	16416075	24187603	2817039
Великобританія	45	9,00%	1569089	2457528	222300
Німеччина	27	5,40%	2288766	2933697	472332
Франція	26	5,20%	1214250	1827024	220928
Китай	21	4,20%	1379877	2536415	210640
Японія	16	3,20%	994211	1308592	125320
Канада	9	1,80%	439783	661945	60340
Австрія	8	1,60%	219920	266644	23222
Нова Зеландія	8	1,60%	233222	422650	40688
Росія	8	1,60%	646246	822672	77992
Швеція	7	1,40%	300733	423616	39500
Італія	6	1,20%	200689	272840	21968
Іспанія	6	1,20%	205897	310012	32992
Швейцарія	5	1,00%	346885	417933	52360
Саудовска Аравія	4	0,80%	310865	370283	77920
Індія	3	0,60%	199257	279702	23416
Нідерланди	3	0,60%	110653	155970	21336
Польща	3	0,60%	81392	130531	13416
Фінляндія	2	0,40%	103047	151547	14584
Ізраїль	2	0,40%	49340	148374	13272
Південна Корея	2	0,40%	298170	338159	27768
Норвегія	2	0,40%	72454	110984	11182
Австралія	1	0,20%	21906	40960	4096
Бельгія	1	0,20%	22731	42485	3984
Бразилія	1	0,20%	64630	72397	6464
Болгарія	1	0,20%	23862	27850	8192
Гонконг	1	0,20%	32325	56672	5600
Ірландія	1	0,20%	25110	28672	2560
Малайзія	1	0,20%	20442	38224	4096
Сингапур	1	0,20%	23482	28128	2400
Словенія	1	0,20%	36752	49152	4096
Південна Африка	1	0,20%	25440	30861	2624
Всього	**500**	**100%**	**27977501**	**40950122**	**4664627**

І варто відмітити значну динаміку зміни лідерів серед країн виробників суперкомп’ютерів. З року в рік змінювався рейтинг держав і технічні характеристики суперкомп’ютерних систем. Якщо країни знаходяться на шляху інтенсивного розвитку і потребують розв’язування актуальних технічних,

соціальних задач іноваційного характеру, то вони прогресують і досягають нових здобутків і переваг у розробці сучасних суперкомп'ютерних систем, інакше вони втрачають здобуті переваги. Ситуація також залежить від наукоємності прдукції, яку виробляє країна, і яким чином вона підтримує розвиток іноваційних технологій. Підтвердженням цього є наступні приклади. Так, в першій редакції рейтингу Top500 (1993 р.) на долю Японії припадало 22,2% усіх суперкомп'ютерів (111 штук), а в 34 редакції (2009 р.) – 3,2% (16 штук). В 1993році в рейтингу суперкомп'ютерів не значились ні Китай, ні Росія, а на вказаному рубежі вони займали 5 та 12 місця. Подібна картина спостерігається і серед виробників, рейтинг яких наведений в таблиці 4.

Відома фірма Cray Inc, на долю виробництва якої в 1993 р. припадало приблизно 41% парку суперкомп'ютерів, на рубежі минулих десятиліть - 3,2% суперЕОМ. Фірми IBM в 1993 р. просто не було в рейтингу, а зараз вона уже багато років знаходиться на перших позиціях.

Українські суперкомп'ютери не можуть тягатися зі світовими лідерами. Здобутки наші надто скромні. І про це не легко казати людині, яка в 1968 році робила українську суперЕОМ-систему спочатку із 4-х, а потім розширену до 8 ЕОМ Дніпро-2, зауваживши, що Дн-2 на той час була повним функціональним аналогом ЕОМ системи IBM-360.

Я це навожу для того, щоб нагадати Вам, що це були українські системи і мали вони не такий вже «борщагівський» вигляд.

Таблиця 4. Рейтинг виробників суперкомп’ютерів

Виробник	К-ть	Доля %	R_{max}Сум (GF)	R_{peak} Сум (GF)	Processor Сум
Hewlett-Packard	209	41,80 %	6339227	10909344	1022764
IBM	185	37,00 %	9722029	14611698	2142560
Cray Inc.	19	3,80 %	4450150	5732805	596315
SGI	19	3,80 %	1834559	2141518	198304
Dell	16	3,20 %	616040	824297	85766
Sun Microsystems	11	2,20 %	1517670	1821631	171442
Appro International	6	1,20 %	458160	569703	71712
Fujitsu	5	1,00 %	318740	353277	33688
Bull SA	5	1,00 %	481080	546316	54576
Hitachi	3	0,60 %	209601	288754	23072
NEC	2	0,40 %	173190	191283	6656
Self-made	2	0,40 %	43800	108482	10240
Dell/Sun/IBM	2	0,40 %	132720	168912	18360
Intel	1	0,20 %	36000	41714	3408
Linux Networx	1	0,20 %	40610	52992	4416
Dawning	1	0,20 %	180600	233472	30720
T-Platforms	1	0,20 %	350100	414419	35360
NEC/Sun	1	0,20 %	87010	163188	31024
DALCO AG Switzerland	1	0,20 %	28990	34560	2880
ClusterVision/Dell	1	0,20 %	41770	48005	4096
Koi Computers	1	0,20 %	37420	56448	6720
Pyramid Computer	1	0,20 %	32800	52032	5420
ACTION	1	0.20 %	38170	49731	5336
ClusterVision/IBM	1	0,20 %	28775	37632	3360
SKIF/T-Platforms	1	0,20 %	47170	60000	5000
Raytheon-Aspen Systems/Appro	1	0,20 %	26730	36601	3440
Lenovo	1	0,20 %	102800	145965	12216
NUDT	1	0,20 %	563100	1206190	71680
DELL\ACS	1	0,20 %	38490	49152	4096
Всього	**500**	**100%**	**27977501**	**40950122**	**40950122**

Далі перемістимось в 2017 рік і наведемо п’ять перших суперкомп'ютерів зі списку Top500 в 2017 році.

Найпотужнішим суперкомп'ютером в світі на 2017 рік є ***Sunway TaihuLight***.

Продуктивність Sunway TaihuLight становить 93 петафлопс.

Sunway TaihuLight, який використовується для кліматичного моделювання і проведення медико-біологічних досліджень, містить 10,65 мільйонів ядер - близько 41 тисячі вузлів. Суперкомп'ютер розроблений Національним дослідницьким центром паралельної обчислювальної техніки і технологій КНР, його розмістили в Національному центрі суперкомп'ютерів в місті Усі провінції Цзяньсу.

Tianhe-2

Tianhe-2 раніше був відомий як MILKYWAY-2 був розроблений Національним університетом технологій оборони Китаю. Став найпотужнішим суперкомп'ютером у світі в 2013 році коли обігнав свого конкурента Titan. На 2017р він стоїть на другому місці. Максимальна швидкість досягає 33,86 petaFLOPS. Така величезна продуктивність завдяки 16 тисячам вузлів, які складаються з Intel Ivy Bridge і Xeon Phi. Китай створив цей суперкомп'ютер спеціально для моделювання різних додатків безпеки.

Piz Daint

Третю позицію займає потужний швейцарський суперкомп'ютер Piz Daint,

який в 2016 році отримав велике оновлення. Модифікація дозволила збільшити продуктивність в три рази, таким чином Piz Daint отримав швидкість 25,3 petaFLOPS. Таким чином зараз цей суперкомп'ютер найпотужніший за межами Азії. Планувався для підключення до Великого адронного колайдеру.

Gyoukou

На четвертому місці знаходиться система Gyoukou, яка належить

Японському агентству науки і технологій з вивчення морських надр (Japan

Agency for Marine-Earth Science and Technology). Суперкомп'ютер має 1250 16-ядерних процесорів Xeon і шину Infiniband EDR, але більшу частину

обчислювальної потужності йому забезпечують 19,84 млн ядер прискорювачів Pezy-SC2. Максимальна стійка продуктивність системи становить 19,14 PFLOPS, а пікова − 28,19 PFLOPS. Дуже суттєвою особливістю Gyoukou стала висока ефективність енергоспоживання, яка становить 14,17 GFLOPS / Вт і в два рази перевершує ефективність сусідів цього суперкомп'ютера по рейтингу Top500.

Gyoukou змістив з раніше займаних місць американський суперкомп'ютер Titan, побудований компанією Cray, а також Sequoia, створений IBM.

Titan

Titan є самим відомим суперкомп'ютером на заході. Знаходиться він в Національній лабораторії Oak Ridge в Теннессі. Довгий час був найпотужнішим суперкомп'ютером у світі, поки Tianhe-2 не обігнав його в 2013 році. Titan перший хто об'єднав в собі процесори AMD Opteron і графічні процесори NVIDIA Tesla, таким чином його продуктивність становить 19,14 PFLOPS, а пікова - 27 petaFLOPS.

Передові держави з великою відповідальністю відносяться до суперкомп’ютерних проблем. Так, наприклад, уряд Японії на 2010 р. для створення національного суперкомп’ютера виділив 253 млн доларів бюджетних грошей і результати такого підходу говорять самі за себе – найвища сходинка в 2020 році.

Станом на 2020 рік, листопад, найпотужнішим визнано суперкомп’ютер Fugaku. Параметри комп’ютера вказано в таблиці 5.

Таблиця 5. Параметри суперкомп’ютера Fugaku

Загальні характеристики	
Виробник	Fujitsu (Японія)
Кількість ядер	7,630,848
Пам’ять	5,087,232 GB
Процесор	A64FX 48C 2.2GHz

Швидкодія	
Rmax (Linpack Performance)	442,010 TFlop/s
Rpeak (Theoretical Peak)	537,212 TFlop/s
Nmax	21,288,960
HPCG [TFlop/s]	16,004.5
Споживання електроенергії	
	29,899.23 kW (Оптимізовано: 26248.36 kW)
Програмне забезпечення	
Операційна система	Red Hat Enterprise Linux
Компілятор	FUJITSU Software Technical Computing Suite V4.0
Math Library	FUJITSU Software Technical Computing Suite V4.0
MPI	FUJITSU Software Technical Computing Suite V4.0

Виробник найпотужнішого станом на листопад 2020 року суперкомп'ютера Fugaku — Fujitsu, Японія. Суперкомп'ютер встановлено в центрі обчислювальних наук "RIKEN" (RIKEN Center for Computational Science, R-CCS) в Японії (Kobe, Japan).

Вище вже вказувалося, що швидкодія суперкомп'ютера вимірюється за результатами стандартного тесту Linpack.

Пакет Linpack — тестова програма для оцінки швидкодії комп'ютера при здійсненні обчислень з плаваючою крапкою (floating point) для розв'язання систем лінійних рівнянь [3]. Linpack був розроблений Джеком Донгаррою (Jack

Dongarra) для надання уявлення про те, скільки часу потребує машина для розв'язання складних задач лінійної алгебри. Результати тестування Linpack можуть бути використані для оцінки швидкості, з якою комп'ютер, найімовірніше, буде запускати програму(и) [5]. Наразі, Linpack складається з наступних тестових пакетів: LINPACK 100, LINPACK 1000, HPLinpack [4].

Перший пакет, LINPACK 100 (програма на Fortran), подібний до оригінального пакету, опублікованого 1979 року. Призначений для матриць 100-го порядку. Для виконання програми не дозволено вносити жодні зміни в її код, навіть у коментарі. Можливе лише оптимізування компіляції для підвищення швидкодії. Зазначимо, що пакет досить малий і негнучкий, що не дозволяє більшості сучасних комп'ютерів досягти їхньої межі продуктивності.

LINPACK 1000 дозволяє забезпечити швидкодію, близьку до граничної, оскільки, крім того, що надано задачу більшого розміру (матриця 1000-го порядку), також дозволено вносити зміни в алгоритм. Єдиними обмеженнями є те, що відносна точність не може бути зменшена, і кількість операцій завжди вважається рівною $2/3n^3 + 2n^2$, де n = 1000.

Пакет HPLinpack було запропоновано як вирішення питання про тести для паралельних комп'ютерів (попередні два пакети для цього не підходять). У HPLinpack розмір n задачі може бути зробленим настільки великим, наскільки це потрібно для оптимізації результатів продуктивності машини. Не зважаючи на використовуваний алгоритм, кількість операцій вважають рівною $2/3n^3 + 2n^2$. Не дозволено використовувати алгоритм Штрассена для швидкого множення матриць, оскільки він спотворює фактичну швидкість виконання. Точність має бути такою, що задовольняє наступну вимогу:

$$\frac{\|Ax - b\|}{\|A\|\|x\|n\epsilon} \leq O(1)$$

,

де є — точність машини, n — розмір задачі, $\|\cdot\|$ — норма матриці, а $O(1)$ відповідає нотації великого О.

Як було вже сказано, метою тестів Linpack є оцінка швидкості вирішення комп'ютером реальних задач. Проте варто зазначити, що оцінка за допомогою одного тесту є спрощенням, оскільки жодна обчислювальна задача не може відобразити загальну ефективність комп'ютерної системи [4].

Детальні характеристики архітектури суперкомп'ютера Fugaku подано в таблиці 6.

Таблиця 6. Архітектура суперкомп'ютера Fugaku [7]

Number of Nodes		
Number of Nodes		158,976 nodes
Peak Performance		
Peak Performance	Normal Mode: 2.0 GHz	• Double Precision (64 bit) 488 Petaflops • Single Precision (32 bit) 977 Petaflops • Half Precision (16 bit) 1.95 Exaflops • Integer (8 bit) 3.90 Exa Exaops
	Boost Mode: 2.2 GHz	• Double Precision (64 bit) 537 Petaflops • Single Precision (32 bit) 1.07 Exaflops • Half Precision (16 bit) 2.15 Exaflops • Integer (8 bit) 4.30 Exaops
Total Memory		4.85 PiB

Total Memory Bandwidth		163 PB/s
Node		
Architecture		Armv8.2-A SVE 512 bit With the following Fujitsu's extensions: Hardware barrier, Sector cache, and Prefetch
Number of computational cores		48 cores
Number of assistant cores		Computational node: 2 cores I/O and computational node: 4 cores
Performance	Normal Mode: 2.0 GHz	DP: 3.072 TFLOPS, SP: 6.144 TFLOPS, HP: 12.288 TFLOPS
	Boost Mode: 2.2 GHz	DP: 3.3792 TFLOPS, SP: 6.7584 TFLOPS, HP: 13.5168 TFLOPS
Memory		HBM2 32 GiB, 1024 GB/s
Interconnect		Tofu Interconnect D (28 Gbps x 2 lane x 10 port)
I/O		PCIe Gen3 x16
Technology		7nm FinFET
Storage System		
1st Layer		Cache for global file system Temporary file systems ● Local file system for compute node ● Shared file system for a job

2nd Layer	Lustre-based global file system
3rd Layer	Cloud storage services (in preparation)
Programming Language and Library	
Compiler	Fortran2008 & Fortran2018 subset
	C11 & GNU and Clang extensions
	C++14 & C++17 subset and GNU and Clang extensions
	OpenMP 4.5 & OpenMP 5.0 subset
	Java
Parallel Programming	XcalableMP
	FPDPS
Script Language	Python + Numpy + Scipy, Ruby
Math Library	BLAS, LAPACK, ScaLAPACK
	SSL II (Fujitsu)
	EigenExa, Kevd, Batched BLAS, 2.5D-PDGEMM
System Software	
OS	Red Hat Enterprise Linux 8
	McKernel
MPI	Fujitsu MPI (Based on OpenMPI), RIKEN-MPICH (Based on MPICH)

File IO	LLIO
	Application-oriented file IO libraries

FLOPS (floating point operations per second, кількість операцій з плаваючою комою за секунду) — одиниця вимірювання швидкодії ЕОМ [6]. Використовується, здебільшого, в наукових обчисленнях з плаваючою точкою. У таких випадках більш точна, ніж IPS (instructions per second). Послуговуються також такими величинами, як kFLOPS (10^3 FLOPS), MFLOPS (10^6 FLOPS), GFLOPS (10^9 FLOPS) та ін.

Продуктивність суперкомп'ютера Fugaku вказана в таблиці 7.

Таблиця 7. Продуктивність Fugaku

Rmax (Linpack Performance)	442,010 TFlop/s
Rpeak (Theoretical Peak)	537,212 TFlop/s
Nmax	21,288,960
HPCG [TFlop/s]	16,004.5

R_{max}: швидкодія для найбільшої задачі, розв'язаної машиною.

N_{max}: розмір найбільшої задачі, розв'язаної машиною.

R_{peak}: теоретична гранична (пікова) швидкодія (theoretical peak performance) для машини.

Далі у таблиці 8 наведено порівняльну характеристику 3-х із 10 найпотужніших суперкомп'ютерів станом на листопад 2020 року (у порядку зменшення потужності зліва направо).

Таблиця 8. Порівняльна характеристика 3-х найпотужніших суперкомп'ютерів станом на листопад 2020 р.

	Supercomputer Fugaku	Summit, IBM Power System AC922	Sierra, IBM Power System AC922
Загальні характеристики			
Виробник	Fujitsu	IBM	IBM / NVIDIA / Mellanox
Країна	Японія	США	США
Кількість ядер	7,630,848	2,414,592	1,572,480
Пам’ять	5,087,232 GB	2,801,664 GB	1,382,400 GB
Процесор	A64FX 48C 2.2GHz	IBM POWER9 22C 3.07GHz	IBM POWER9 22C 3.1GHz
Швидкодія			
Rmax (Linpack Performance)	442,010 TFlop/s	148,600 TFlop/s	94,640 TFlop/s
Rpeak (Theoretical Peak)	537,212 TFlop/s	200,795 TFlop/s	125,712 TFlop/s
Nmax	21,288,960	16,473,600	11,902,464
HPCG [TFlop/s]	16,004.5	2,925.75	1,795.67
Споживання електроенергії			
	29,899.23 kW (Optimized: 26248.36 kW)	10,096.00 kW (Submitted)	7,438.28 kW (Submitted)

Програмне забезпечення			
Операційна система	Red Hat Enterprise Linux (RHEL)	RHEL 7.4	RHEL
Компілятор	FUJITSU Software Technical Computing Suite V4.0	XLC, nvcc	IBM XLC
Math Library	FUJITSU Software Technical Computing Suite V4.0	ESSL, CUBLAS 9.2	ESSL, CUBLAS 9.2
MPI	FUJITSU Software Technical Computing Suite V4.0	Spectrum MPI	IBM Spectrum MPI

Далі приведені технічні характеристики суперкомп’ютера НТУУ КПІ (таблиця 9).

Таблиця 9. Техніні характеристики Cluster

Computing nodes			
	Number of computing nodes	Processors	Number of processor cores

	44	Intel Xeon E5440 (2.83 GHz, RAM – 8 Gb)	4
	68	Intel Xeon 5160 (3.00 GHz, RAM – 8 Gb)	2
Total	112		624
Performance			
Max	7 TFlops		
Linpack test	5.7 TFlops		
Hardware			
Disk space	6 Tb		
Software			
Operating system	CentOS release 5.2		

Уточнюючі:

Суперкомп'ютер складається з двох підсистем.

Перша, яка працює під керуванням ОС Linux, містить 44 узла на базі чотирьохядерних процесорів Intel Xeon E5440 та 8 ГБ ОЗУ та 68 вузлів на основі двоядерних процесорів Intel Xeon 5160 та 4 ГБ ОЗУ. Усі вузли двопроцесорні. Внутрішня підсистема під управлінням ОС Windows Server 2008 HPC включає 16 двопроцесорних вузлів на базі чотирьохядерних процесорів Intel Xeon E5440 та 8 ГБ ОЗУ. Пікова продуктивність суперкомп'ютера складає 7 TFLOPS (5,7 TFLOPS на тестових завданнях). При цьому використані мережа обміну даними InfiniBand й операційна система Linux CentOS release 5.2.

Однією з особливостей кластеру НТУУ «КПІ» є наявність апаратного прискорювача операцій. У рамках третьої модернізації обчислювального комплексу НТУУ «КПІ» створено обчислювальний модуль на базі ПЛІС (FPGA). Результати для вибраних алгоритмів показують доцільність використання цього типу обчислювальних модулів для виконання низки ресурсоємних задач.

Спільна пам’ять процесорів	12•1012 байт або 12 Тб

Архівна пам’ять	20 Тб

Суперкомп'ютер побудований компанією «Юстар»

Джерела

1. http://www.top500.org
2. https://insidehpc.com/hpc-basic-training/what-is-hpc/
3. http://www.netlib.org/utk/people/JackDongarra/faq-linpack.html#_Toc27885709
4. https://en.wikipedia.org/wiki/LINPACK_benchmarks
5. https://searchdatacenter.techtarget.com/definition/Linpack-benchmark
6. https://en.wikipedia.org/wiki/FLOPS
7. https://www.fujitsu.com/global/about/innovation/fugaku/specifications/index.html

1. 3. **Етапи розвитку суперкомп’ютерних технологій**

Приблизно до кінця 80-х років минулого століття вдавалося будувати суперкомп'ютери в рамках фоннейманівської архітектури.

Термін «архітектура комп'ютера» був введений фірмою IBM при розробці обчислювальних систем серії IBM 360 і застосований до тих засобів, які може використовувати програміст під час написання програм на рівні машинних команд. Під цим терміном розуміли структуру комп'ютера з погляду програміста, так звану логічну архітектуру, що є поняттям архітектури у вузькому розумінні. Воно охоплює формати команд, форми зображення даних, методи адресації й керування операціями введення- виведення. В 1946 році Джон фон Нейман описав архітектуру комп'ютера ЕНІАК і ця архітектура дістала назву його імені. Це був опис реалізованого Преспером Екертом та Джоном Моучлі універсального комп'ютера ЕНІАК , створеного в Принстонському університеті, часом її іще називають принстонською архітектурою. Джон фон Нейман був рядовим членом цієї команди і писав звіт про виконану роботу.

Головні особливості архітектури комп'ютера Джона фон Неймана:

> Інформація в комп'ютері ділиться на команди і дані.
> Команди вказують комп'ютеру, які дії і над якими операндами виконувати.
> Послідовність команд, за якою виконується алгоритм вирішення задачі, назива¬ють програмою .
> Весь набір виконуваних комп'ютером команд називають системою команд ком¬п'ютера.
> Дані - це числа і закодовані символи, які використовуються командами як операнди. Одні команди для інших також можуть бути операндами .
> Команди і дані представлено двійковим кодом.
> Немає відмінностей в представленні команд і даних. Наприклад, двійкове число 100011011100 може бути як командою , так і даним .

> Команди і дані зберігаються в одній пам'яті.
> Команди і дані зберігаються в пам'яті за відповідними адресами.
> Пам'ять має довільну адресацію, тобто в кожному такті можна звернутися до до¬вільної її комірки.
> Пам'ять є лінійною. Її адресу кодують двійковим кодом, починаючи від молодшої.

В основу роботи цього комп'ютера покладено принцип програмного керування. Тоб¬то функціонування цього комп'ютера здійснюється потактово за вказівкою команд програми. Програма разом з даними , які підлягають обробці , спочатку записується до основної пам'яті. В кожному такті, залежно від типу архітектури комп'ютера, виконується команда або частина команди - мікрокоманда. Команда вказує тип виконуваної опера¬ції та місце розміщення операндів і результату операції. Для виконання однієї команди необхідно провести наступні дії: записати до програмного лічильника адресу команди, зчитати із основної пам'яті команду за вмістом програмного лічильника, провести де¬шифрування команди з метою її розпізнання , визначити адреси комірок пам'яті, в яких знаходяться операнди та до яких мають бути записані результати, зчитати ці операнди з пам'яті та подати в арифметико-логічний пристрій для опрацювання, виконати операцію над операндами та записати результати до основно ї пам'яті. Таким чином проводиться виконання всіх команд програми. По закінченню в основній пам'яті будуть знаходитися результати виконання програми.

Отже, за архітектурою фон Неймана комп’ютер – це:

• пристрій введення,
• центральний процесор,
• арифметико-логічний пристрій
• запам'ятовуючий пристрій,
• пристрій виведення.

Ускладнюючи внутрішній пристрій процесора, можна було змусити його виконувати приблизно (або навіть точно) такі ж послідовності машинних команд, що і «звичайний» процесор, але набагато швидше. Іншими словами, фоннеймановська архітектура залишалася адекватною формою організації зростання кількості транзисторів, придатних для інтеграції в єдину обчислювальну систему.

На цьому етапі:

— програміст майже не задумувався над тим, для суперкомп'ютера чи «звичайного» комп'ютера він пише програму;

- розробник обладнання суперкомп'ютера розроблював "суперпроцесор";

— суперкомп'ютер таким чином був звичайною фоннеймановською машиною, оснащеною «суперпроцесором».

Сама можливість значно прискорювати процесор порівняно зі зразками, які випускаються серійно, удосконалюючи лише його внутрішній пристрій, пояснювалася дефіцитом транзисторів, які використовуються при виготовленні «звичайного» процесора. Розробники суперкомп'ютерів могли мати у своєму розпорядженні набагато більше транзисторів, ніж розробники машин, які

серійно випускаються, що дозволяло їм застосовувати технічні рішення, просто недоступні для останніх. «Суперпроцесор» працював швидше «звичайного» процесора, зрештою, саме за рахунок того, що на його виготовлення вдалося витратити набагато більше транзисторів і можна було використовувати біль нові транзистори, крупносерійне виробництво яких ще тільки запроваджувалося. Для комп'ютерів масового вжитку це було недоступно. Можливість використовувати більш швидкодіючі транзистори із когорти новосторених, масовий серійний випуск яких тільки започатковувався, був немаловажним фактором. Це обумовлювалось невеликими потребами таких елементів для реалізації суперкомп'ютерів, так як вони виготовлялися одиничними екземплярами. І все робилося саме так.

Я добре пам'ятаю той час, коли я їздив в місто Зеленоград Московської області в центр мікроелектроніки Радянського Союзу на завод «Мікрон» НДІ Молекулярної електроніки, на завод «Ангстрем» НДІ Твердого тіла і привозив цілі валізи дослідних зразків високочастотних транзисторів, які вказані заводи використовували при проведенні їх приймальних випробувань. В серійних виробах такі елементи не могли бути використані, бо їх ще не було навіть в відповідних каталогах, а ми їх вже використовували при створенні суперпроцесору. Про Зеленогад -чисте виробниц-заводи в лісі – за 80 км від Москви на північний Захід

Приблизно в кінці 80-х років минулого століття розробники надвеликих інтегральниз схем змогли розмістити на одному кристалі кремнію мільйон транзисторів (це приблизно технологічний рівень процесора Intel 80486). Маючи у своєму розпорядженні таку кількість елементів, розробники процесорів отримали можливість прискорити практично все, що можна було прискорити в принципі. Звичайно, спроби «добути» додаткову швидкодію на мікрорівні тривають і зараз — на кристалі мікропроцесора Pentium-2, наприклад, транзисторів уже мільярд. Але вже тоді, на мільйонному рівні, стало очевидно, що багаторазового (у десятки та сотні разів) прискорення роботи, яка виконується в середньому за один такт, не отримати. Епоха «супер-процесорів» скінчилася - відтепер кожен процесор є одночасно і «звичайним», і «супер», суттєвої різниці немає. Єдиним методом виготовлення суперкомп'ютеру стало виготовлення паралельної обчислювальної системи, тобто машини, в якій багато процесорів спільно працюють над одним завданням. Такі машини будувалися з мікросхем, які серійно випускаються, зокрема, мікропроцесорів, але на оригінальній схемотехнічній базі (шафи, плати, лінії зв'язку).

Звертаю Вашу увагу, що це відбулося наприкінці 80-х років минулого століття, а в 1968 році на Київському заводі обчислювальних машин був створений такий «суперкомп'ютер» спочатку із 4-х ЕОМ Дніпро-2 (повний функціональний аналог американської системи IBM-360), а в подальшому із 8-и ЕОМ Дн-2. Вважайте, це робили ви. Ми знали не більше, ніж ви зараз. Запитань було більше, ніж відповідей. Я слухав лекції Глушкова. Дуже вражало його щире зізнання, що він часом не знає того, про що ми його запитували. Зараз такого не скажуть. Зараз все всі знають.

Цей техенологічний переворот вважають першою суперкомп'ютерною революцією.

На цьому етапі:
- програміст, щоб отримати суперкомп'ютерний рівень швидкодії, був поставлений перед необхідністю писати спеціальні – паралельні – програми;
- розробник обладнання суперкомп'ютеру переключився зі створення «суперпроцесорів» на розробку засобів об'єднання багатьох стандартних, серійно випускаємих мікропроцесорів в єдину систему;
— таким чином, суперкомп'ютер став уже не фоннейманівською машиною, а паралельною обчислювальною системою.

Можливість отримувати суперкомп'ютер за допомогою розробки власних засобів інтеграції стандартних мікропроцесорів в єдину систему існувала доти, поки серійновипускаємі засоби інтеграції, тобто обладнання локальних мереж, були розвинені порівняно слабо і для інтеграції процесорів «на суперкомп'ютерному рівні» не годилися. В міру вдосконалення обладнання локальних мереж та інтерфейсів їх підключення до комп'ютерів, які серійно випускаються, і ця ситуація змінилася. Приблизно наприкінці 90-х років серійновипускаєме мережеве обладнання дозволило використовувати як суперкомп'ютер звичайну локальну мережу, компактно розташовану і оснащену відповідним програмним забезпеченням. Технічний та економічний сенс розробки та виготовлення власних плат та шаф для інтеграції стандартних мікропроцесорів у єдину систему зник. Переважна більшість суперкомп'ютерів стала будуватися за технологією кластерів виділених робочих станцій, тобто збиратися методами системної інтеграції з готових, серійновіпускаємих комп'ютерів загального призначення, плат і комутаторів локальних мереж.

Цей переворот називають другою суперкомп'ютерною революцією.

На цьому етапі:
— для програміста, порівняно з попереднім етапом, нічого принципово не змінилося, крім того, що таких програмістів стало набагато більше, як і самих суперкомп'ютерів;
- розробник суперкомп'ютерного обладнання змушений був стати системним інтегратором - розроблювати стало нічого, всі використовувані компоненти та засоби їх інтеграції стали випускатися великими серіями;
- суперкомп'ютер, в результаті, з паралельної обчислювальної системи на оригінальній схемотехнічній базі став кластером виділених робочих станцій.

Звернемо увагу до значення слова «виділених» у цьому випадку. Сама по собі ідея використовувати локальну мережу як суперкомп'ютер значно старша за другу суперкомп'ютерну революцію. У проміжку між першою та другою суперкомп'ютерними революціями локальні мережі бурхливо розвивалися, в тім рахунку в організаціях, зацікавлених у високопродуктивних обчисленнях. Звичайно, робилися спроби використовувати локальні мережі загального призначення, наприклад, під час нічних простоїв, для паралельних обчислень. Такі локальні мережі за своїми комунікаційними здатностями значно поступалися «справжнім» суперкомп'ютерам, але на деяких, що не потребують особливо інтенсивних комунікацій між вузлами, паралельні програми вдавалося успішно виконувати. Саме тоді з'явилася назва «обчислювальний кластер» і перші варіанти програмного забезпечення, які дозволяли

використовувати локальну мережу в цій якості. Це програмне забезпечення було спрямоване на паралельне використання локальної мережі в обох режимах: в основному - як локальної мережі загального призначення, і лише іноді - як обчислювального кластеру, тобто обладнання не виділялося спеціально для використання в режимі обчислювального кластера.

Друга суперкомп'ютерна революція, як ми тільки що бачили, полягала в тому, що при необхідності виділити обладнання для використання виключно як обчислювальний кластер, це обладнання стало оформлюватися також у вигляді локальної мережі. На відміну від кластерів попереднього етапу, такі кластери стали забезпечуватися зовсім особливим програмним забезпеченням, яке часто виключає використання вузлів як машин загального призначення (на вузлі сучасного обчислювального кластеру електронну пошту не надсилають, і наукові статті до публікації не готують).

Щоб розрізняти два способи організації паралельних обчислень на устаткуванні локальних мереж і, відповідно, два принципово різних способи організації програмного забезпечення, «нові» кластери стали називати кластерами виділених робочих станцій, а старі, відповідно, кластерами невиділених робочих станцій. Програмні технології організації кластерів невиділених робочих станцій існують і сьогодні.

Далі, якщо явно не обумовлено зворотнє, називатимемо кластерами саме кластери виділених робочих станцій — продукт другої суперкомп'ютерної революції. Технологія кластерів виділених робочих станцій панує в суперкомп'ютерній галузі, фактично, визначаючи її вигляд.

На підході третя суперкомп'ютерна революція. Правильніше було б сказати, що вона вже почалася. Масштаб змін, які внесе її завершення у світ технологій паралельної обробки даних, буде безпрецедентним. Саме це заважає нам охарактеризувати її суть «у двох словах», як було зроблено у разі першої та другої суперкомп'ютерних революцій. Залишимо це захоплююче оповідання до наступних лекцій, а поки зупинимося на цій точці зору: сьогодні (вчора?) увесь світ рахує, в основному, на кластерах.

Література

1 http7/www. intel. com. Мережевий ресурс.
2 http 7/www, amd. com. Мережевий ресурс.

1. 4 **Способи об’єднання багатьох процесорів в єдину систему**

Вже в 60-тих роках минулого століття комп'ютери розділилися на машини серійного виробництва і машини рекордної продуктивності, які випускаються окремими екземплярами або малими серіями. Останні стали називати суперкомпьютерами, а способи їх побудови і використання - технологіями високопродуктивних обчислень. До деяких пір ці технології розвивалися в рамках фоннейманівських принципів, але в кінці 80-х, коли це перестало бути можливим, суперкомп'ютери стали паралельними обчислювальними системами, а технології високопродуктивних обчислень стали невіддільними від технологій паралельної обробки даних.

У своєму розвитку паралельні суперкомп'ютери пройшли два чітко помітних етапа.

На першому етапі вони будувалися, в основному, як системи з великого числа інтегрованих в єдину систему послідовних машин, або ж послідовних машин з невеликими, чітко окресленими паралельними розширеннями, такими, як команди векторної обробки даних. На цьому етапі існує багато різних способів побудови системи зі складових її вузлів, але вузлом в будь-якому випадку є «звичайний» або «майже звичайний» комп'ютер. Даний етап був тривалим. Завершення першого і перехід до другого етапу обумовлений тим, що для перетворення доступної неймовірної кількості електронних елементів в реально досяжну обчислювальну швидкодію вже не досить архітектури з так чи інакше взаємодією фоннеймановських вузлів. Потрібно міняти архітектуру («конструкцію») самого вузла.

Процес появи відповідних технологій почався. Майже напевно його результати по своїй революційності і несподіванки перевершать всі мислимі і немислимі очікування і прогнози.

Отже, після завершення епохи «суперпроцесорів» єдиним способом виготовлення суперкомп'ютера стало виготовлення паралельної обчислювальної системи, тобто машини, в якій багато процесори спільно трудяться над одним завданням. Такі машини будувалися з серійних мікропроцесорів, але на оригінальній схемотехнічній базі (шафи, плати, лінії зв'язку).

На цьому етапі:

- програміст, щоб отримати суперкомп'ютерний рівень швидкодії, був поставлений перед необхідністю писати спеціальні - паралельні програми;
- розробник обладнання суперкомп'ютера переключився зі створення «суперпроцесорів» на розробку засобів об'єднання багатьох стандартних, серійних мікропроцесорів в єдину систему;
- таким чином, суперкомп'ютер був вже не Фоннеймановською машиною, а паралельною обчислювальною системою.

В кінці 90-х серійне мережеве обладнання дозволило використовувати в якості суперкомп'ютера звичайну локальну мережу, компактно розташовану і оснащену відповідним програмним забезпеченням. Технічний і економічний сенс розробки і виготовлення власних плат і шаф для інтеграції стандартних мікропроцесорів в єдину систему пропав. Переважна більшість суперкомп'ютерів стало будуватися за технологією кластерів: збиратися методами системної інтеграції з готових серійних комп'ютерів загального призначення, плат і комутаторів локальних мереж [1].

На цьому етапі:

- для програміста, в порівнянні з попереднім етапом, нічого принципово не змінилося, крім того, що таких програмістів стало багаторазово більше, як і самих суперкомп'ютерів;
- розробник суперкомп'ютерного обладнання змушений був стати системним інтегратором - розробляти стало нічого, всі використовувані компоненти і засоби їх інтеграції стали випускатися великими серіями;

- суперкомп'ютер, в результаті, з паралельної обчислювальної системи на оригінальній схемотехнической базі став кластером виділених робочих станцій.

Способи об'єднання багатьох процесорів в єдину систему

Багатопроцесорні обчислювальні комплекси прийнято ділити на мультипроцесори - системи із загальною, або розподіленою (між процесорами) пам'яттю, і мультикомп'ютери - системи з самостійних комп'ютерів, кожен - зі своєю, локальною пам'яттю, об'єднаних комунікаційною мережею. Мультикомп'ютери частіше називають системами з розподіленою (між процесорами) пам'яттю, або MPP- системами (Massively Parallel Processing, масово-паралельна обробка), або просто системами без загальної пам'яті. На даний момент така класифікація не може бути визнана достатньою. Як клас мультипроцесорів, так і клас мультикомп'ютерів підрозділяються на підкласи, які якісно розрізняються за способами інтеграції процесорів. В результаті способи інтеграції процесорів в єдину обчислювальну систему утворюють не тільки ієрархію, а й безперервний спектр, від способів з найбільш сильними взаємозв'язками між процесорами з класу мультипроцесорів до способів з найбільш слабкими взаємозв'язками з класу мультикомп'ютерів (рис. 1).

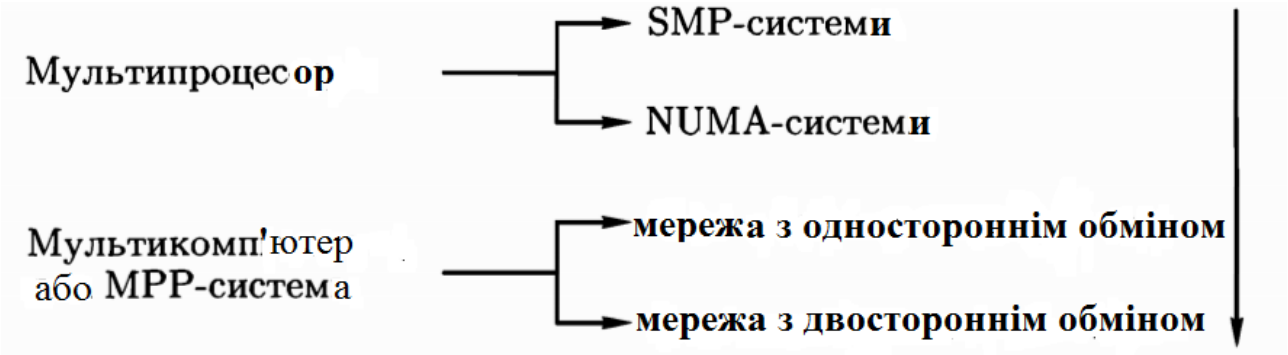


Рис. 1. Ієрархія і спектр способів інтеграції процесорів в обчислювальну систему (вертикальна стрілка праворуч показує напрямок ослаблення взаємозв'язків між процесорами).

Розглянемо спектр способів інтеграції в традиційному порядку: спочатку опишемо крайні випадки, а потім заповнимо проміжок між ними. Для стислості тут і далі будемо говорити про якісно різних способах інтеграції процесорів як про «сильних» і «слабких».

На самому «сильному» краю спектра у верхній частині малюнку знаходяться SMP-системи (Symmetric Multi Processing). У цих мультипроцесорах вся наявна в системі оперативна пам'ять загальна, і вся вона так само доступна всім процесорам.

Самий «слабкий» край спектра в нижній частині малюнку представлений MPP-системами на базі мереж двостороннього обміну даними. У цих системах для передачі даних з пам'яті одного процесора в пам'ять іншого необхідні спеціальні, явно записані в програмі дії на обох кінцях каналу. Той, кому потрібні «чужі» дані, повинен виконати запит на прийом даних, а той, у кого ці дані зберігаються - запит на їх передачу.

Тільки при такому спільному бажанні обох учасників обміну дані можуть бути передані від одного процесора до іншого (звідси назва: «мережі двосторонніх обмінів»).

Тепер зупинимося на проміжних по «силі» рішеннях.

Другий за силою спосіб інтеграції процесорів - NUMA- системи (Non-Uniform Memory Access). Це системи, в яких пам'ять загальна, але не обов'язково вся, а головне - не всі області пам'яті доступні всім процесорам на рівних правах. Спільна пам'ять в такій системі поділена (зазвичай - на блоки). Кожен з таких блоків належить конкретному процесору, тобто звернення до нього з цього процесора відбуваються швидко. Всі інші, крім «свого», блоки загальної пам'яті також доступні процесору для прямої адресації звичайними командами звернення до пам'яті, але виконуються ці команди набагато повільніше, ніж при доступі в «свій» блок. Залежно від конкретної реалізації, слово «набагато» в попередній фразі може означати «вдвічі повільніше», а може - і «в 300 разів повільніше».

Нарешті, третій по «силі» спосіб інтеграції процесорів представлений MPP-системами на базі мережі односторонніх обмінів. У цих системах для передачі даних з пам'яті одного процесору в пам'ять іншого досить записати відповідний запит в програмі лише одного процесору. Наприклад, один процесор може «насильно» покласти деякі дані в зазначене ним місце пам'яті іншого процесору. Інший процесор фізично не бере участі в цьому обміні - всю роботу робить його мережевий адаптер, який отримав дані і запит на їх запис по мережі від мережевого адаптеру - ініціатора обміну. У підсумку, кожен процесор працює з пам'яттю будь-якого іншого процесора приблизно так само, як з власним жорстким диском, самостійно контролюючи весь процес передачі даних.

Короткий опис класифікації способів, якими процесори інтегруються в єдину обчислювальну систему, завершено. Тепер подивимося на неї обчислювальну систему очима програміста.

Легко виявити, що перераховані способи інтеграції, дійсно, утворюють безперервний спектр в сенсі зручності запису і ефективності реалізації доступу до «чужих» даних.

В разі SMP-системи немає самого поняття «чужих» даних. Всі дані для всіх процесорів «свої». У NUMA-системі доступ до «чужих» даних записати так же просто, як до «своїх», але потрібно враховувати в структурі програми, що доступ цей - набагато повільніший. У MPP-системі на базі мережі односторонніх обмінів доступ до «чужих» даних не просто набагато повільніший, ніж до «своїх», а й записується в програмі спеціальним чином, в вигляді запиту на обмін, подібно запиту на обмін з жорстким диском або іншим пристроєм зовнішньої пам'яті. Нарешті, в MPP-системі на базі мережі двосторонніх обмінів доступ до «чужих» даних не просто записується спеціальним чином, а й вимагає «у відповідь люб'язності» від партнера - «господаря» даних.

Природно припустити, що цим чотирьом способам повинні відповідати чотири якісно різних способи запису доступу до «чужих» даних в прикладній програмі, тобто чотири сімейства систем паралельного програмування - мов та /

або бібліотек. Надалі буде показано, що це дійсно так. Більш того, способи запису доступу до «чужих» даних в програмі, як і породивші їх способи побудови багатопроцесорної обчислювальної системи, бувають «сильними» і «слабкими». Краще всього складати програму способом, «сила» якого відповідає «силі» використовуваної апаратури, але так буває далеко не завжди. «Слабкі» способи легко реалізуються на «сильній» апаратурі, але не повністю використовують її можливості. «Сильні» способи реалізуються на «слабкій» апаратурі з великими труднощами, часто - вельми неефективно. Про це ми поговоримо при розгляді моделей і технологій паралельного програмування.

Тепер подивимося на класифікацію способів інтеграції процесорів в систему очима розробника устаткування.

SMP-система - найбільш зручний для програміста, але найбільш дорогий в технічній реалізації спосіб. Звичайно, при такому способі програміст ділить між процесорами тільки роботу, але не дані, причому ділити роботу можна дуже дрібними частинами (порядка десятків команд), що буває досить зручно. Однак, крім високої вартості, такі системи погано масштабуються: навіть в дуже дорогих системах число поєднуваних процесорів не дотягує до сотні. Якість об'єднання часто залишає бажати кращого: чим більше процесорів, тим більше вони схильні конкурувати за доступ до загальної пам'яті, взаємно сповільнюючи роботу один одного.

Відзначимо, що всі популярні останнім часом багатоядерні процесори є, з логічної точки зору, SMP-системами.

MPP-системи значно дешевше в розрахунку на один процесор, прекрасно масштабуються (до десятків тисяч вузлів), але на них важче програмувати. Ділити між процесорами доводиться не тільки роботу, а й дані. Причому роботу доводиться «нарізати» великими частинами (порядку сотень тисяч команд і більше), а передачі даних, розташованих в одному процесорі, але необхідних для обчислень в іншому, доводиться явно записувати в програмі. Друга суперкомп'ютерна революція посилила і закріпила панування цієї технології: з появою можливості будувати MPP-системи взагалі без спеціальних апаратних розробок цінова прірва між ними і SMP-системами порівняльного розміру стала ще набагато ширшою, ніж раніше. Весь прогрес в здешевленні і зростанні доступності суперкомп'ютерів, досягнутий другою суперкомп'ютерною революцією, припав на долю MPP-систем.

Досить часто вузлом MPP-системи буває не поодинокий однопроцесорний комп'ютер, а SMP-система порівняно невеликого розміру. Ще зовсім недавно середні позиції вибудуваного спектра були заповнені слабо. Серед мультипроцесорів було мало NUMA-систем, і вартість їх часто практично не відрізнялася від вартості SMP-систем. Єдиною перевагою (до речі, дуже важливою) була значно більш висока масштабованість.

З іншого боку, переважна більшість MPP-систем були системами на базі мереж двостороннього обміну даними. За роки, що минули після другої суперкомп'ютерної революції, саме в цих проміжних позиціях спектру спостерігався стійкий і значний прогрес. NUMA- системи росли в розмірах і дешевшали, залишаючись, проте, суперкомп’ютерами на оригінальній схемотехнічній базі (тобто не кластерами).

При цьому в класі мережевого обладнання для кластерів виділених робочих станцій стало з'являтися все більше рішень, які підтримують на апаратному рівні односторонні обміни даними. NUMA-систему можна розглядати як «удосконалену» мережу односторонніх обмінів, в якій односторонній обмін запускається автоматично, на апаратному рівні, при виконанні команди, адресуючу «повільну», тобто «чужу» пам'ять. Очевидно, існування цих двох споріднених, але не рівноцінних проміжних класів - NUMA-систем і мереж одностороннього обміну даними - можливо лише в разі, якщо системи «менш зручного» класу набагато дешевші і доступніші. До недавнього часу це так і було. NUMA-системи можна було будувати виключно на основі оригінальної схемотехніки рівня плат і шаф, причому із застосуванням високовартісних замовних великих інтегральних схем , в той час як мережі односторонніх обмінів були кластерами на базі серійного обладнання.

Зміни в архітектурі і технології випускаємих великими серіями материнських плат привели, фактично, до стирання цієї межі. Це веде не тільки до злиття двох проміжних класів представленого спектру, але і до часткового поглинання новоутвореним класом «звичайних» кластерів на базі мереж двостороннього обміну.

Альтернативні архітектури. Класифікація Флінна

До сих пір мовчазно вважалося, що паралельна обчислювальна система, в будь-якому випадку, будується з універсальних (фоннеймановських) процесорів. Це не завжди так. Наприклад, векторний процесор - це також паралельна не фоннеймановска архітектура, але отримана не об'єднання в одну систему багатьох фоннеймановских процесорів, а іншим способом - додаванням до універсального процесору команд роботи з векторами [2, 3]. Ми не будемо зараз спеціально обговорювати питання векторних обчислень. Однак зазначимо, що вузол як SMP-, так і MPP-системи може бути і векторним процесором. Також, як зазначалося раніше, вузол MPP-системи може бути SMP- або NUMA-системою, розмір якої помітно менше розміру установки в цілому.

Той факт, що паралельна обчислювальна система, взагалі кажучи, не повинна будуватися на базі саме фоннеймановських процесорів, означає можливість (і необхідність) класифікації таких систем, образно кажучи, по «виду паралелізму». Таких класифікацій існує багато, і найвідомішу з них - класифікацію Флінна - хотілося б згадати. Ця класифікація будується на понятті єдиності або, навпаки, множинності одночасно оброблюваних обчислювальною системою потоків команд і даних. Оскільки є два типи потоків (команди і дані), і варіантів для кожного - також два (один потік або багато), всього існує чотири варіанти структури обчислювальної системи.

1. Один потік команд, один потік даних - SISD (Single Instruction, Single Data). Це - фоннеймановський комп'ютер.
2. Один потік команд, багато потоків даних - SIMD (Single Instruction, Multiple Data). Це - векторний комп'ютер, процесор якого може, наприклад, однією командою скласти покомпонентно два масиви чисел.
3. Багато потоків команд, багато потоків даних - MIMD (Multiple Instruction, Multiple Data). У цей клас потрапляють всі згадані нами вище багатопроцесорні

обчислювачі на базі фоннеймановських процесорів, незалежно від способу об'єднання цих процесорів між собою - SMP, NUMA, MPP.
4. Багато потоків команд, один потік даних - MISD (Multiple Instruction, Single Data). Деякі дослідники комп'ютерної архітектури вважають, що окремі, експериментальні обчислювальні установки, відомі їм, могли б претендувати на включення в цей клас. Більшість авторів вважає, що цей клас порожній, і навряд чи буде заповнений в доступному для огляду майбутньому. Іноді практикується віднесення до цього класу систоличних масивів [3] і навіть векторноконвейерних структур (має характер деякої «термінологічної натяжки»).

Література

1. А.Н. Андреев, В. В. Воеводин, С.А.Жуматий. Кластеры и суперкомпьютеры — Открытые системы / — М., 2000. -№ 5-6- С. 9-14
2. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. Учебное пособие – М.: Издательство Московского университета, 2011. – 563с .
3. Воеводин В. В. Параллельные вычисления. — СПб. : БХВ- Петербург, 2002. — 608 с.

2 ***Програмне забезпечення суперкомп'ютера***.

2. 1. Введення в програмне забезпечення суперкомп'ютера.

Програмне забезпечення суперкомп'ютера.

Розповідати про склад і призначення програмного забезпечення суперкомп'ютерів найкраще на конкретному прикладі. Виберемо модельну програму, сплануємо «на папері» її паралельну реалізацію, а потім - подивимося, якого роду програмне забезпечення нам буде потрібно, щоб виконати цю паралельну реалізацію на суперкомп'ютері взагалі, і на кластері робочих станцій - зокрема.

1. Рішення двовимірної крайової задачі для рівняння теплопровідності методом Якобі

Фізика завдання: розглянемо однорідний прямокутний паралелепіпед («цеглина»), до бічних граней якого щільно притулені нагрівні елементи нескінченної маси і заданої температури. В міру прогріву «цегли» в ній встановиться деякий розподіл температури, який нам і потрібно знайти. Будемо розглядати двовимірне наближення задачі. Це означає, що цегла передбачається дуже високою, і нас буде цікавити розподіл температури в її горизонтальному зрізі, досить віддаленому як від верхньої, так і нижньої граней. Для стислості будемо називати цю задачу «завданням про прогріванні цегли». Підкреслимо, що ні фізична, ні обчислювально-математична сторона цієї прекрасно вивченої задачі нас не цікавить - ми розглядаємо її як приклад заданої і не обговорюваної обчислювальної процедури, для якої треба побудувати паралельну реалізацію [2].

Дане завдання є краєвою задчею для рівняння в частинних похідних (рівняння теплопровідності), яку можна наблизити завданням на рівномірній прямокутної сітці. Сіткову задачу, в свою чергу, можна вирішувати ітераційним методом -

ми виберемо метод Якобі. Чисельний метод, до якого ми приходимо, рухаючись згаданим шляхом, дуже простий.
Є двовимірний масив F, значення елементів якого - нормалізовані значення температури у вузлах сітки, «накинутої» на зріз цегли. По краях масиву знаходяться значення температури нагрівачів - граничні умови. Вони в процесі розрахунку не змінюються. З усіма іншими значеннями чинимо так:
- для кожного осередку обчислюємо нове значення як середнє арифметичне значень чотирьох її сусідів;
- після того, як нові значення обчислені для всіх осередків, замінюємо старі значення новими, попутно обчислюючи максимум абсолютної величини відхилення нових значень від старих;
- повторюємо ці два кроки до тих пір, поки максимум абсолютної величини відхилення не стане менше заданого порогу малості.
Для спрощення наших модельних розглядів виключимо з алгоритму обчислення відхилень - будемо просто виконувати фіксоване число ітерацій.
Текст послідовної (для одного процесора) реалізації даної обчислювальної процедури представимо у вигляді:

```
include <stdio.h>
/***/
#define MX 640
#define MY 480
#define NITER 10000
#define STEPITER 100
       static float  f[MX][MY];
       static float df[MX][MY];
/***/
       int main( int argc, char **argv )
{
       int i, j, n, m;
       FILE *fp;
/***/
       printf( "Solving heat conduction task
                on %d by %d grid\n", MX, MY );
       fflush( stdout );
```

```
/* Initial conditions: */
        for ( i = 0; i < MX; i++ )
         {
        for ( j = 0; j < MY; j++ )
         {
          f[i][j] = df[i][j] = 0.0;
          if       ( (i == 0)
                   || (j == 0) ) f[i][j] = 1.0;
          else if ( (i == (MX-1))
                   || (j == (MY-1)) ) f[i][j] = 0.5;
         }
       }
/* Iteration loop: */
        for ( n = 0; n < NITER; n++ )
         {
        if ( !(n%STEPITER) )
             printf( "Iteration %d\n", n );
/* Step of calculation starts here: */
         for ( i = 1; i < (MX-1); i++ )
          {
           for ( j = 1; j < (MY-1); j++ )
            {
             df[i][j] = ( f[i][j+1] +
                          f[i][j-1] + f[i-1][j] +
                          f[i+1][j] ) * 0.25 - f[i][j];
            }
          }
         for ( i = 1; i < (MX-1); i++ )
          {
           for ( j = 1; j < (MY-1); j++ )
            {
             f[i][j] += df[i][j];
            }
          }
       }
 /* Calculation is done, F array is a result: */
```

```
    fp = fopen( "progrev.dat", "w" );
    for ( i = 1; i < (MX-1); i++ )
          fwrite( f[i]+1, MY-2, sizeof(f[0][0]),
                  fp );
    fclose( fp );
    return 0;
}
```

2. Паралельна реалізація «на папері»

Найпростіша паралельна реалізація очевидна. Поділимо масиви на горизонтальні смуги (групи рядків) рівної висоти, по числу процесорів. Нехай кожен процесор обробляє свою смугу на кожній ітерації (рис. 2.1.).

Під словом «обробляє» тут розуміються ті рядки, елементам яких процесор привласнює нові значення. Конкретний зміст слова «поділимо» (рядки між процесорами) залежить від того, чи є в системі спільна пам'ять.

Якщо наша машина - SMP-система, то паралельна реалізація побудована; якщо це NUMA-система, то ми її побудували, за умови, що нам вдасться розташувати кожну смугу масивів F і dF в пам'яті, «своїй» для оброблюючого ці смуги процесора. В обох цих випадках весь масив F, зображений на малюнку, розташовується в загальній пам'яті, доступній всім процесорам. При цьому кожен процесор, знаючи власний номер, виконує цикл обходу рядків не всього масиву, а тільки своєю смуги: нульовий процесор модифікує рядки з 1-го по 4-ий, перший - з 5-го по 8-й, і т. Д . Межі зміни змінної циклу по рядках у кожного процесора свої. Нульовий і 13-й рядки масиву містять граничні умови і не обробляються. Обробка 12 рядків поділена між процесорами порівну, кожному дісталося по чотири рядки.

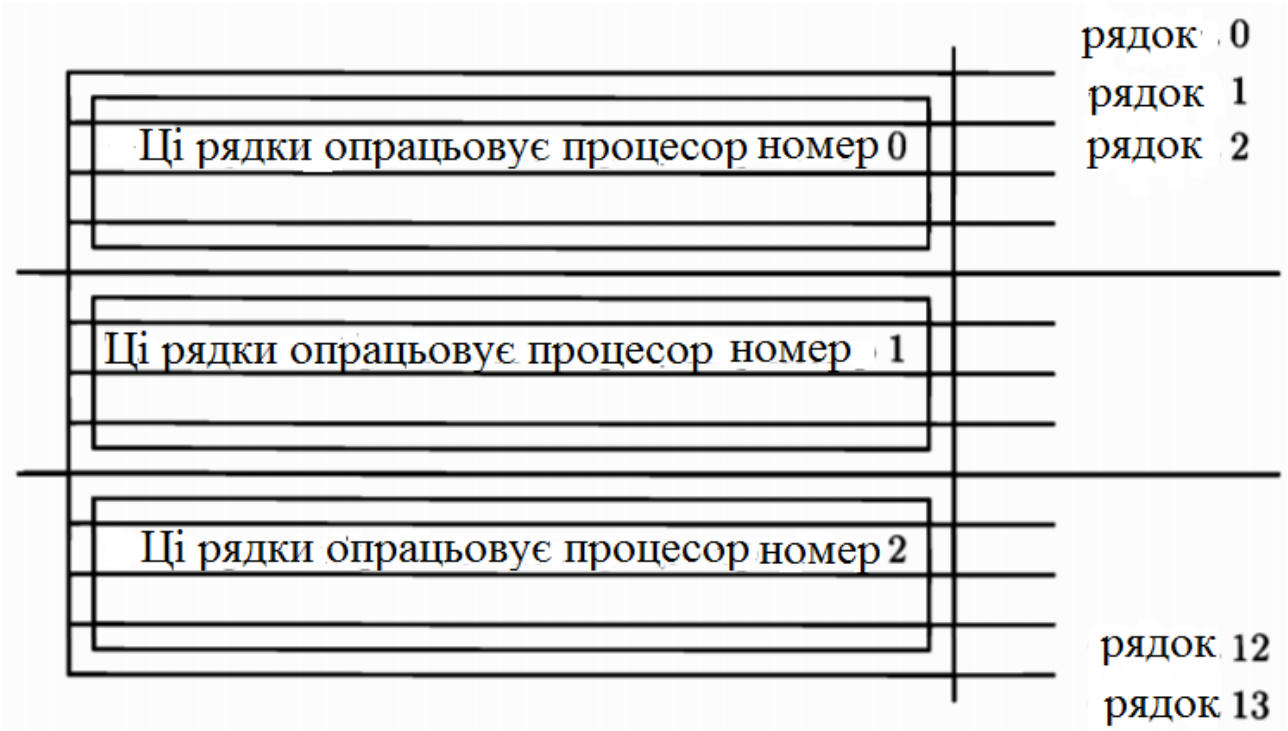


Рис. 2.1. Масив F з 14 рядків, поділений між трьома процесорами (рядки 0 і 13 - граничні умови)

Зараз нас більше цікавить випадок, коли розрахунок виконується на MPP-системі (наприклад, на обчислювальному кластері). Загальної пам'яті у процесорів кластеру немає, отже, немає і єдиного масиву F (точніше, він є, але не в тексті програми, а тільки в голові програміста). Замість загального для всіх процесорів масиву F, у кожного процесора тепер є свій масив F, розміром з «діставшоїся» цьому процесору смуги. Кожен процесор проходить цей масив цілком - межі зміни змінної циклу по рядках у всіх процесорів одні й ті ж. Обробка кожного з таких часткових масивів в кожному процесорі повинна бути побудована так, щоб до кінця розрахунку у всій цій сукупності часткових масивів виявилися в точності ті ж значення, які виявилися б у єдиному масиві F при розрахунку на одному процесорі або на системі із загальною пам'яттю. Наприклад, якщо спочатку виконати розрахунок на одному процесорі і записати підсумковий масив F в файл на диску, а потім повторити розрахунок на чотирьох процесорах і записати в файл на диску один за одним вміст масивів F кожного з процесорів, в порядку їх номерів, то файли повинні вийти ідентичними. У цьому випадку завдання побудови паралельної реалізації не можна вважати закінченою, оскільки незрозуміло, що робити з краями смуги масиву F. Справді, обчислюючи перший і останній рядки смуги, процесор, взагалі, змушений скористатися крайніми рядками сусідніх смуг, а ці рядки розташовані на сусідніх процесорах. Отже, необхідно відводити смуги з «запасом» в один рядок на початку і один в кінці, а перед початком кожної ітерації передбачити в програмі передачу даних з країв смуги кожного процесора в «запасні» рядки сусідніх процесорів, щоб кожен з процесорів мав «свіжу» копію «сусідських» країв, нараховану минулої ітерації. Зрозуміло, під час запису кінцевих результатів на диск «запасні» рядки записувати не слід (рис. 2.2.).

Рис. 2.2. Локальна порція масиву F на першому процесорі

Для нульового і останнього процесорів, які не мають «сусіда зверху» і «сусіда знизу», відповідно роль копій сусідських країв виконують рядки граничних умов. Їх не треба нізвідки копіювати, оскільки вони в процесі розрахунку не змінюються.

Тепер паралельну реалізацію «на папері» можна вважати побудованою і для випадку кластеру. Далі будемо, якщо явно не зазначено протилежне, говорити тільки про кластери (точніше, про MPP-системи).

3. Паралельна реалізація на практиці

Спробуємо реалізувати на практиці те, що просто і логічно виглядає на папері. З одного боку, є кластер, наприклад, з восьми процесорів. Це комп'ютери, кожен зі своєю ОС, і вони пов'язані мережею. З іншого боку, є розуміння, яка програма повинна виконуватися на кожному з них, і загальне уявлення про те, як (на змістовному рівні) ці програми повинні обмінюватися даними одна з одною.

Чи означає це, що ми повинні написати вісім вихідних текстів, розкласти по вузлах кластера вісім виконуваних модулів, а потім, швидко перебігаючи від клавіатури до клавіатури, запустити все це на восьми комп'ютерах? І як ці виконувані модулі потім зв'яжуться між собою для обміну даними? Питання риторичне.

Перш за все, розберемося з гіпотетичною необхідністю написання восьми вихідних текстів. Деякі системи паралельного програмування передбачають можливість написання окремих вихідних текстів програми для різних вузлів, але в більшості випадків все ж чинять інакше.

Програмісту пропонується написати текст програми для *i*-го вузла, де значення *i* стане відомим лише під час виконання програми. В даному випадку це не тільки не важко, але і дуже бажано. З одного боку, тексти програм для всіх вузлів, очевидно, дуже схожі, з іншого - не хотілося б щось міняти в тексті, якщо ми захочемо виконати програму не на восьми, а на десяти або сорока вузлах кластера.

Написана нами програма для узагальненого вузла номер *i* називається гілкою паралельної програми, а вся сукупність гілок, кожна з яких запускається на своєму вузлі - паралельною програмою.

Запуск зазначеної користувачем кількості гілок паралельної програми на відповідній кількості вузлів виконується з єдиної точки, наприклад, з самого першого вузла, або взагалі зі спеціальної керуючої машини, яка не є вузлом. Досить часто команда запуску - це командний файл (скрипт), який просто виконує команду дистанційного (по мережі) запуску виконуваного файлу гілки на кожному із задіяних вузлів кластера.

Однак просто запустити гілку дистанційно на зазначеному вузлі недостатньо. У процесі запуску необхідно виконати ще, як мінімум, дві дії:

- якимось чином повідомити кожній з запущених гілок її порядковий номер (те саме значення *i*) і загальне число запущених гілок;
- подбати про те, щоб запущені гілки могли пересилати одна одній повідомлення, користуючись номерами в якості адрес.

Розглянемо найпростіший (і досить рідко застосовуваний, але цілком працездатний) варіант вирішення першої з проблем.

У більшості операційних систем (і вже тим більше - в Unix, що зазвичай використовується на вузлах обчислювального кластеру) є поняття командного рядка (тє, що передається при запуску виконуваного файлу в якості аргументів у функцію *main*). Домовимося, що згаданий вище скрипт - команда запуску - додає до наявних аргументів командного рядка кожної гілки ще два, які представляють собою власний номер гілки і загальне число запущених гілок. Знаючи про цю домовленість, деяка ініціалізуюча функція, викликана користувачем на початку роботи гілки, могла б отримати ці значення і запам'ятати всередині себе, попутно скорегувавши командний рядок, який в програмі, можливо використовується. Наприклад, в бібліотеці MPI це звернення до ініціалізуючої функції виглядає приблизно так:

```
int main( int argc, char *argv )
{
.......
MPI_Init( &argc, &argv );
.......
```

Коли програма користувача звернеться до іншої бібліотечної функції - «дізнатися власний номер» - запомнене таким чином число буде їй повідомлено. Наприклад, в MPI:

```
MPI_Comm_rank( MPI_COMM_WORLD, &my_number );
```

// my_number тепер містить номер гілки.

Конкретний спосіб передачі значень, насправді, не важливий. Цікавлячі нас два числа можуть передаватися через командний рядок, через тимчасові файли або якось ще, в командному рядку додаткові параметри можуть бути першими або останніми. Важливо тут те, що гілка паралельної програми повинна бути запущена з дотриманням деяких правил, причому правила ці дотримуються на двох сторонах: на боці команди запуску і на стороні бібліотеки функцій, якими користується гілка паралельної програми.

Нехай, наприклад, у нас є два варіанти бібліотеки MPI, виготовлених різними авторами, в першому з яких службові параметри вирішено було додавати на початок командного рядка, а в другому - в кінець (або взагалі передавати зовсім іншим способом). Тоді немає сенсу збирати виконуваний файл гілки з бібліотекою від першого варіанту, а запускати його - командою запуску від другого. Те ж стосується і передачі складніше організованих даних, які необхідні для початкового налаштування обміну повідомленнями між гілками по ходу розрахунку. Конкретних способів організації та передачі таких даних може бути багато, і програміст зовсім не зобов'язаний їх знати, але передача цих даних з команди запуску і трактування переданих даних всередині програми повинні бути узгоджені.

Багато (хоча і не всі) системи паралельного програмування мають на увазі наявність доступу процесорів, на яких запускаються гілки паралельної програми, до загальної файлової системи. Справді, передбачені, скажімо, в Unix засоби дистанційного запуску виконуваних файлів передбачають передачу по мережі запиту на запуск, але зовсім не самого виконуваного файлу, який потрібно запустити. Якщо виконуваний файл, який потрібно запустити

дистанційно, доступний на віддаленій стороні через загальну файлову систему, процес запуску сильно полегшується.

Тепер, коли ми уявляємо собі в загальних рисах як труднощі, що виникають при необхідності запустити паралельну програму на кластері, так і підходи до їх подолання, можна приступати до формального перерахування необхідних компонентів програмного забезпечення обчислювального кластера.

Зробимо це в наступному.

Литеретура

1. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. Учебное пособие – М.: Издательство Московского университета, 2011. – 563с .
2. Лацис А. О. Параллельная обработка данных : учеб. пособие для студ.

вузов / — М.: Издательский центр «Академия», 2012. — 336 с. — (Университетский учебник. Сер. Прикладная математика и информатика).

3. Воеводин В. В. Параллельные вычисления. — СПб. : БХВ- Петербург, 2002. — 608 с.

2. 2. *Загальний склад програмного забезпечення суперкомп'ютера*

Наведемо перелік компонентів програмного забезпечення обчислювального кластера.

1. Кожен вузол кластера повинен бути оснащений операці йною системою ОС вузла, включаючи необхідну мережеву підтримку, можливо, спеціального, застосовуваного тільки в обчислювальних кластерах, мережевого обладнання та / або спеціальних мережевих протоколів.

2. Бажано, щоб всі вузли мали доступ до загальної мережевої файлової системи.

3. Для запису передачі даних з вузла у вузол необхідно передбачити комунікаційну бібліотеку (зазвичай - реалізація MPI), яка часто спирається на спеціальну мережеву підтримку.

4. З комунікаційною бібліотекою повинна бути інтегрована система запуску паралельних програм. Це - саме той компонент, який «розкидає» (автоматично, за допомогою стандартних мережевих можливостей) команду запуску, або сам виконуваний модуль по всім комп'ютерам, піклуючись при цьому про те, щоб,

почавши виконуватися, ці програми встановили зв'язок одна з одною, і могли обмінюватися даними.

5. Якщо кластер використовується як машина колективного користування, з системою запуску інтегрується система черг і контролю доступу.

6. В свою чергу, над базовими комунікаційними бібліотеками типу MPI нерідко надбудовуються, для зручності розробки прикладних програм, більш високорівневі бібліотеки і мови паралельного програмування.

7. Якщо, крім усього іншого, передбачено використання кластеру як частини розподіленого обчислювального ресурсу, об'єднуючого багато, незалежно адмініструючихся суперкомп'ютерів, доведеться додати ще й засоби метакомп'ютингу (Grid-технологій) [2]. Ці засоби неминуче зачіпають системи запуску, черг і контролю доступу, але можуть включати в себе і програмні надбудови над комунікаційної бібліотекою.

В подальшому викладі звернемо увагу на те, які зі згаданих технологій при побудові кластеру запозичуються в готовому вигляді зі світу комп'ютерних технологій масового застосування, а які розроблюються спеціально для обчислювальних кластерів.

Програмістська модель передачі даних

При записі в програмі обмінів повідомленнями між гілками можуть використовуватися різні програмістські моделі передачі даних. Найбільш часто використовується модель рандеву, при якій дані передаються від передавача до приймача тоді і тільки тоді, коли передавач видає запит на передачу даних, а приймач, відповідно, на прийом. Порядок спрацьовування передавача і приймача не суттєвий - синхронізація, тобто «очікування запізнившогося», відбувається автоматично, всередині комунікаційної бібліотеки.

Крім усіляких високорівневих програмних надбудов, які полегшують запис рандеву в програмі, є також базові моделі, принципово відмінні від рандеву, наприклад, модель односторонньої передачі повідомлень [1,2, 3,].

Технології паралельної обробки даних без використання суперкомп'ютерів.

Можливість організувати якомога більш тісну взаємодію між гілками паралельної програми в процесі спільної роботи гілок над вирішенням єдиного завдання принципово важлива для програміста, і розробники обчислювальних кластерів намагаються оснастити свої вироби як можна більш швидкими мережами. Однак, існує (хоча і не є переважаючим в процентному відношенні) досить великий клас завдань, при вирішенні яких взаємодія між гілками паралельної програми взагалі не потрібна. Точніше, вона потрібна на початку і в кінці виконання завдання, але обсяг її малий у порівнянні з об'ємом самого розрахунку. Це так званий варіантний рахунок, коли необхідно виконати дуже багато однакових розрахунків з різними вихідними даними, а інтеграція отриманих результатів порівняно не складна. Оскільки взаємодія між гілками не потрібна, гілки можуть виконуватися в різному темпі, наприклад, на однопроцесорних машинах з різною швидкодією. Інтеграція результатів відкладається «на потім», коли порахованих варіантів накопичиться багато. На відміну від переважаючих суперкомп'ютерних технологій, які позначаються терміном HPC (High Performance Computing - високопродуктивні обчислення), технології варіантного рахунки прийнято позначати терміном HTC (*High Throughput Computing* - обчислення з високою пропускною здатністю, високоінтенсивні обчислення). Можливо, що для таких розрахунків немає потреби будувати суперкомп'ютер - досить розробити програмні засоби відбору невикористаної обчислювальної потужності з мереж загального призначення. Це - абсолютно особлива область, яку ми не будемо розглядати в нашому курсі спеціально. Тут мова йде про завдання метакомп'ютингу [2] в її найбільш загальному вигляді. Періодично робляться спроби, наклавши деякі обмеження на дисципліну відбору потужності, все ж скористатися технологіями метакомп'ютингу для вирішення завдань в дисципліні HPC, тобто організувати виконання «кластерних», паралельних програм не на виділеному обладнанні, а на системах програмного відбору потужності. В загальному вигляді це складна технологічна задача.

Литеретура

1. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. Учебное пособие – М.: Издательство Московского университета, 2011. – 563с .
2. Лацис А. О. Параллельная обработка данных : учеб. пособие для студ.
   вузов / — М.: Издательский центр «Академия», 2012. — 336 с. — (Университетский учебник. Сер. Прикладная математика и информатика).
3. Воеводин В. В. Параллельные вычисления. — СПб. : БХВ- Петербург, 2002. — 608 с.

2. 3. ***Моделі і технології паралельного програмування***

Структури паралельних обчислювальних систем (ПОС)

Способи взаємодії паралельних задач в паралельних обчислювальних системах визначається організацією їх пам’яті.[1]

Розг Паралельні обчислювальні системи зі спільною пам’яттю

Паралельна обчислювальна система (ПОС) зі спільною пам’яттю (англ. shared memory) складається з процесорів та пам’яті, до якої мають доступ всі процесори. Будь-яка взаємодія між процесорами відбувається через змінні в спільній пам’яті.

ПОС зі спільною пам’яттю можна розділити на дві категорії: з рівномірним доступом до пам’яті (UMA, uniform memory access) та з нерівномірним доступом до пам’яті (NUMA, non-uniform memory access). В UMA системах з рівномірним доступом до пам’яті (рис. 1) час доступу будь-якого процесору до будь-якої комірки пам’яті однаковий.

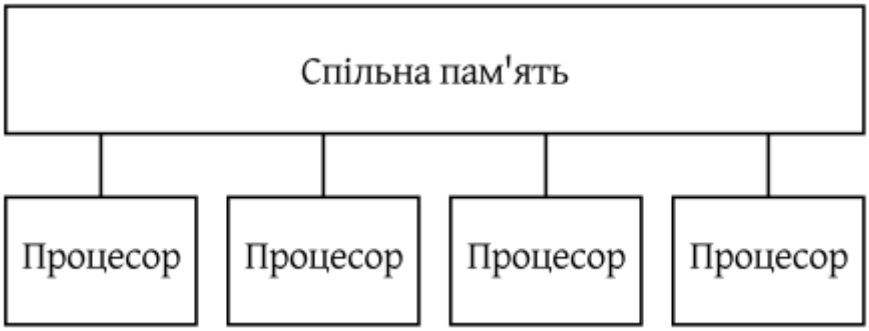


Рис. 1. Спрощена структурна схема ПОС зі спільною пам’яттю UMA

NUMA системи з нерівномірним доступом до пам’яті складаються з кількох UMA систем з рівномірним доступом до пам’яті , з’єднаних системою зв’язків. В NUMA системи з нерівномірним доступом до пам’яті кожен процесор також має доступ до всіх комірок пам’яті.
Але до одних модулів пам’яті процесор під’єднаний безпосередньо, а до інших - через систему зв’язків. Тому час доступу до різних комірок пам’яті відрізняється.
Але з точки зору програміста, в програмному коді доступ до «віддалених» комірок пам’яті нічим не відрізняється від доступу до «близьких» комірок - система зв’язків в NUMA системи з нерівномірним доступом до пам’яті є прозорою для програміста.
Конструктивно NUMA системи системи з нерівномірним доступом до пам’яті звичайно представляють собою систему з однією материнською платою, що містить кілька сокетів для багатоядерних процесорів. Кожен з багатоядерних процесорів безпосередньо під’єднаний до кількох модулів пам’яті. Процесори з’єднані між собою за допомогою ліній зв’язку на материнській платі. Перевагою ПОС з СП є швидка передача даних між процесорами.

Недоліки ПОС з СП:

• Малий потенціал розширюваності системи при додаванні процесорів: це призводить до збільшення кількості читань/записів в пам’ять, а пам’ять має фіксовану пропускну здатність.
• Системи з СП з великою кількістю процесорів є відносно дорогими (частково через попередній пункт).

На рис.2 показана спрощена структурна схема ПОС зі спільною пам’яттю NUMA.

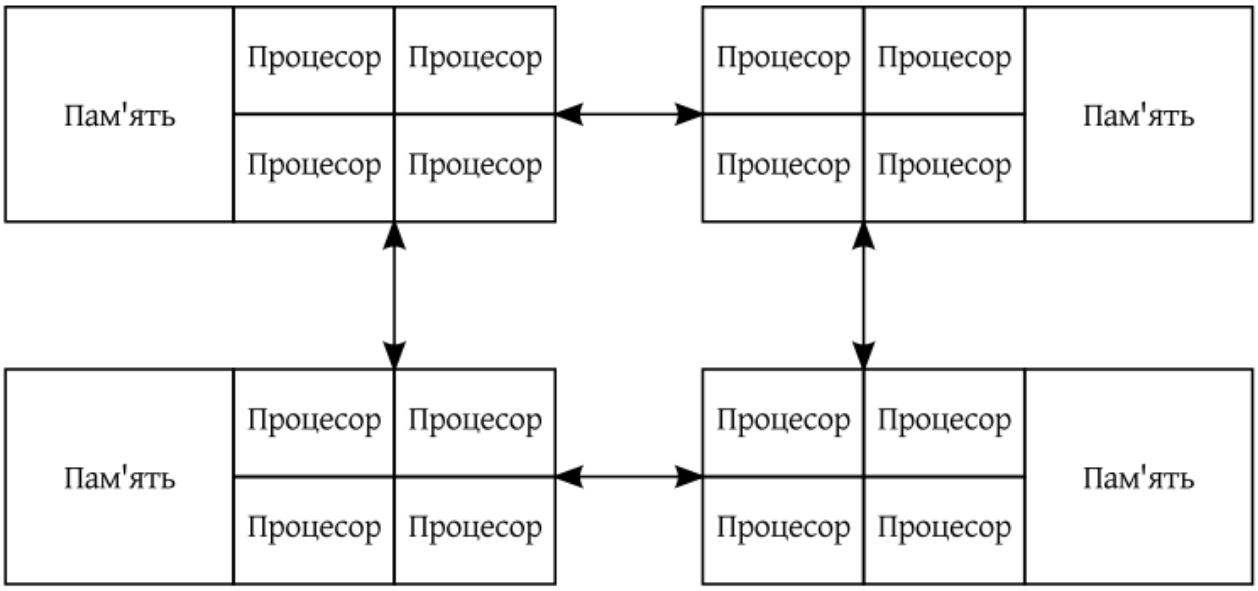


Рис. 2. Спрощена структурна схема ПОС зі спільною пам’яттю NUMA

Паралельні обчислювальні системи з локальною пам’яттю**.**

ПОС з локальною пам’яттю (англ. Distributed memory) складаються з обчислювальних вузлів, з’єднаних системою зв’язків. Обчислювальний вузол складається з процесора та пам’яті. Доступ до пам’яті в обчислювальному вузлі має виключно процесор того ж самого вузла. Будь-яка взаємодія між вузлами здійснюється через систему зв’язку. Спрощена структурна схема ПОС з локальною пам’яттю показана на рис.3.

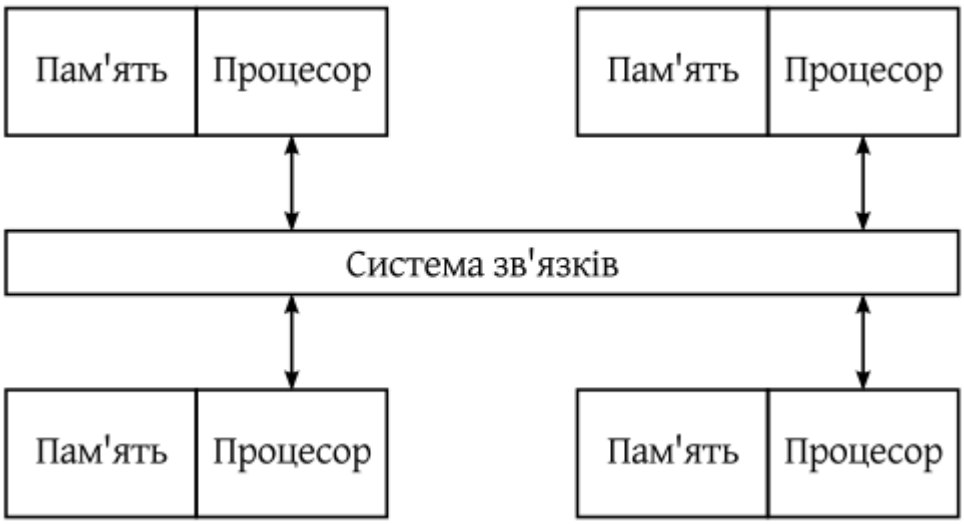


Рис 3. Спрощена структурна схема ПОС з локальною пам’яттю

Конструктивно ПОС з ЛП звичайно представляють собою однопроцесорні обчислювальні вузли, з’єднані за допомогою комп’ютерної мережі. Так як час передачі даних між вузлами для багатьох задач є критичним, то звичайно використовують спеціальні високопродуктивні мережі (наприклад, Infiniband).

Для деяких задач (з малою кількістю даних для передачі) можна використовувати звичайні мережі
(наприклад, Ethernet).

Переваги:

• Пам’ять розширюється автоматично з додаванням процесорів.

• Не витрачається час на підтримку когерентності кешів. Тому доступ до пам’яті є більш швидким порівняно з ПОС з СП, а процесори - більш простими.

• Можливість побудови з неспеціалізованих компонентів (звичайні однопроцесорні одноядерні системи, з’єднані мережею Ethernet).

Недоліки:

• Час передачі даних між вузлами значно більший, ніж час передачі даних за допомогою спільної пам’яті в ПОС з СП.

ПОС гібридної архітектури.

ПОС гібридної архітектури мають одночасно і спільну, і локальну пам’ять. Такі системи складаються з обчислювальних вузлів, які являють собою системи зі спільною пам’яттю. Обчислювальні вузли з’єднані між собою за допомогою системи зв’язку, при цьому процесори одного вузла не мають доступу до пам’яті інших вузлів. Взаємодія між процесорами в межах вузла відбувається через спільну пам’ять, а між процесорами в різних вузлах - через систему зв’язку. Спрощена структурна схема ПОС гібридної архітектури зображена на рис. 4.

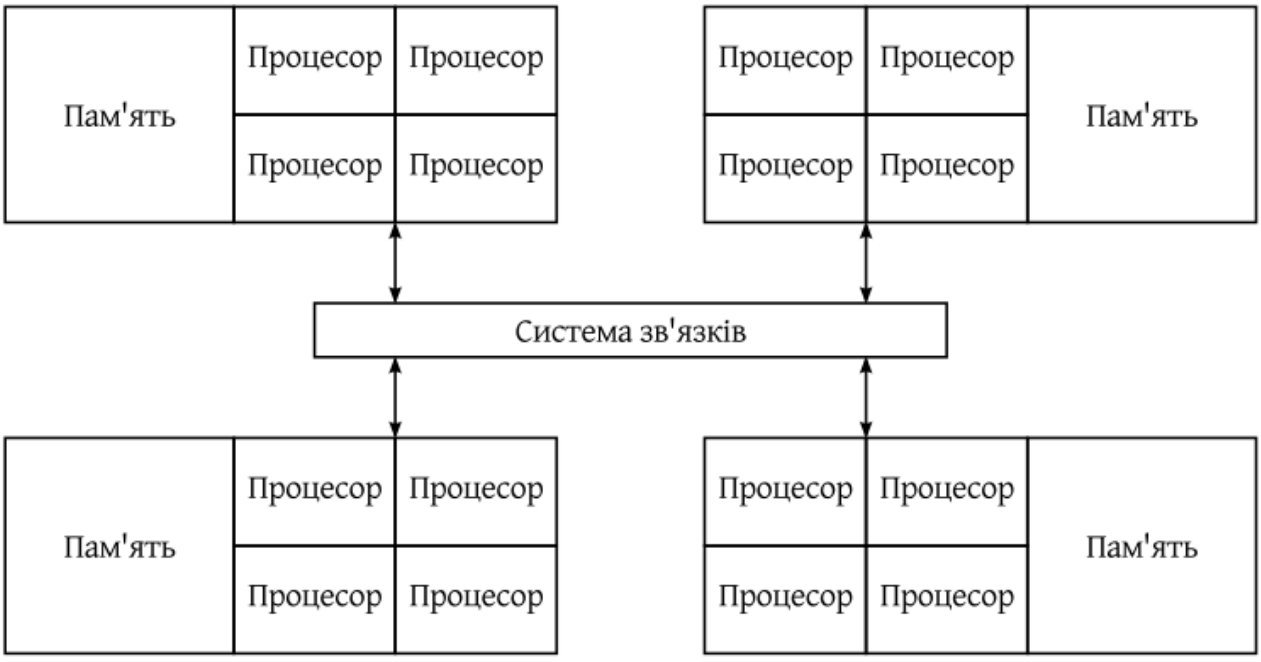


Рис.4. Спрощена структурна схема ПОС гібридної архітектури

До ПОС гібридної архітектури відносяться майже всі сучасні кластерні системи. В ролі вузлів зі спільною пам’яттю виступають багатоядерні багатопроцесорні системи, а в ролі системи зв’язків - комп’ютерна мережа. Окремі машини самі по собі можна розглядати як паралельні системи зі спільною пам’яттю. Але для передачі даних між машинами в кластері необхідно явно використовувати систему зв’язків.
Переваги та недоліки - ті ж самі, що й у систем зі спільною пам’яттю та у систем з локальною пам’яттю.

Моделі взаємодії задач

Інструменти паралельного програмування (мови та бібліотеки) приховують від програміста деталі архітектури конкретної паралельної обчислювальної системи.
Програміст працює з деякою *моделлю* взаємодії задач, яка визначає множину допустимих дій.

Модель взаємодії повинна забезпечувати виконання наступних дій:
• передача даних;
• синхронізація.

Існують дві основні моделі взаємодії задач: модель, яка базується на спільних змінних, та модель, яка базується на відправці повідомлень.

Ці моделі розглянемо в одній з наступних лекцій.

Литература

1. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. Учебное пособие – М.: Издательство Московского университета, 2011. – 563с .

2. Лацис А. О.Параллельная обработка данных : учеб. пособие для студ. вузов / — М.: Издательский центр «Академия», 2012. — 336 с. — (Университетский учебник. Сер. Прикладная математика и информатика).

3. Воеводин В. В. Параллельные вычисления. — СПб. : БХВ- Петербург, 2002. — 608 с.

4. Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко– К., 2011. – 154 с.

5. Жуков І.А. Паралельні та розподілені обчислення [Текст] / І.А. Жуков, О.В. Корочкін. – Київ: «Корнійчук», 2014. – 284 с.

2. 4. ***Моделі взаємодії задач***

Інструменти паралельного програмування (мови та бібліотеки) приховують від програміста деталі архітектури конкретної паралельної обчислювальної системи. Програміст працює з деякою *моделлю* взаємодії задач, яка визначає множину допустимих дій.

Модель взаємодії повинна забезпечувати виконання наступних дій:

• передача даних;

• синхронізація.

Існують дві основні моделі взаємодії задач: модель, яка базується на спільних змінних, та модель, яка базується на відправці повідомлень.

В **моделі, яка базується на спільних змінних**, задачі працюють в одному адресному просторі та виконують передачу даних та синхронізацію за допомогою спільних (глобальних) змінних. Ця модель найкраще підходить для паралельних обчислюальних систем зі спільною пам'яттю. В паралельних обчислюальних систем з локальною пам'яттю теоретично цю модель можна застосувати, але на практиці для паралельних обчислюальних систем з локальною пам'яттю немає реалізацій, що працювали б ефективно для всіх класів задач.

Перевага даної моделі полягає в тому, що спільний адресний простір є більш простим з точки зору програміста в багатьох ситуаціях. Наприклад: дані, які необхідні кільком задачам, доступні їм зразу ж після того, як ці дані з'явились в спільній пам'яті (наприклад, після введення з файлу). Інший приклад: передача даних між задачами не потребує їх копіювання. З іншого боку, спільний адресний простір створює і основний недолік цієї моделі: програміст відповідальний за коректний доступ до даних з кількох задач.

Наприклад: якщо дані, необхідні кільком задачам, вводяться з файлу, то програміст має забезпечити, щоб задачі не зчитували ці дані зі спільної пам'яті до тих пір, доки дані не введені повністю.

Взаємодія процесів

Взаємодія поцесів пов'язана з двома основними завданнями:

- комунікацією процесів;

- синхронізацією процесів.

Комунікація процесів передбачає обмін даними між процесами і при цьому інформація передається від одного процесу до іншого в будь-якому напрямку.

Згадаємо, що *процес* – абстрактне поняття, яке включає опис певних дій, пов'язаних з виконанням програми в комп'ютерній системі[1]. При цьому процес оформлюється так, щоб система керування процесами в операційній системі (ОС) могла ефективно перерозподіляти ресурси в системі (процесори, пам'ять, пристрої вводу-виводу, файли та інше). Процес характеризується власним набором ресурсів і виділеною для нього ділянкою оперативної пам'яті. Поняття процесу з'явилося в багатозадачних операційних системах і сьогодні є

фундаментальним в сучасних ОС. В ОС процес пов'язаний з кожною прикладною або системною програмою, яка виконуються в комп'ютерній системі. Це дозволяє системі керування процесами ОС, яка розміщується в ядрі ОС, ефективно маніпулювати процесами за допомогою спеціального блоку керування процесом (БКП) або PCB - Process Control Block) [1]. Блок керування процесом – динамічна структура даних, яка містить основну інформацію щодо процесу:

- ім'я процесу;
- поточний стан процесу;
- пріоритет процесу;
- місце розміщення процесу в пам'яті;
- ресурси, що пов'язані з процесом та ін.

На рис. 1 представлено адресний простір процесу. Він містить три розділи: *текстовий* (для коду програми), *інформаційний* (для даних), *стековий* (для стеків програми)

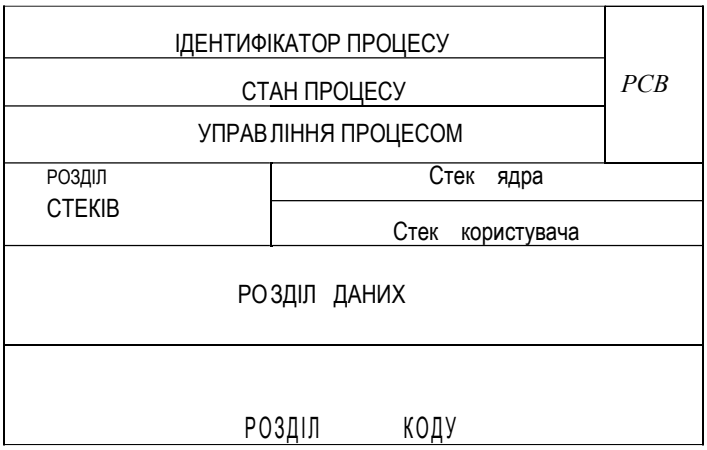


Рис. 1. Логічна структура процесу

Велика кількість програм, які виконуються в системі, визначають перший рівень паралелізму - рівень процесів. Це - багатозадачність. Другій рівень паралелізму - рівень потоків. Він визначає паралелізм всередині однієї програми. Це багатопотоковість.

Паралельне програмування пов'язано з використанням другого рівня паралелізму. Потоки - це процеси, які виконуються в паралельній програмі. Такі процеси отримали назву легкі процеси (lightweight processes). Традиційні процеси в ОС отримали назву важких процесів (heavyweight processes). Для позначення легких прцесів в різних мовах і бібліотеках паралельного програмування використовують різні терміни: потік (Java, C, C#), задача (Ада, MPI), процес. Разом з терміном *процес* використовують також терміни *задача, потік, нить* в залежності від конкретної реалізації процесу.

Важкі та легкі процеси мають багато спільного - ідентифікатор, стан, пріоритет та інші атрибути.

В чому полягає відмінність важких (тяжких) і легких процесів? Перш за все системні затрати в ОС на створення, підтримку і керування легкими процесам (потоками) значно нижчі. Всі ресурси тяжкого процесу поділяються його потоками. Потоки не володіють ресурсами, вони конкурують за їх використання. Стеки потоків знаходяться у стековому

розділі тяжкого процесу. Потік на відміну від тяжкого процесу *не має* свого адресного простору. Всі потоки тяжкого процесу знаходяться в одному адресному просторі.

Під час виконання тяжких процесів у комп'ютерній системі або звичайному комп'ютері відбувається постійне перемикання з одного процесу на інший. Те ж саме відбувається і для легких процесів. Але час, який витрачається на перемикання для легких процесів, менший, ніж для важких. Ще одна відмінність важких та легких процесів полягає в тому, що легкі процеси постійно *взаємодіють,* оскільки є частинами однієї програми (так звані *тісно зв'язані* процеси), в той час, як важкі процеси взаємодіють рідко (*слабко зв'язані* процеси).

Стан процесу. Керування процесами

Керування процесом включає виконання деяких операцій над ним, дозволяючи процесу пройти визначені стани (рис. 2.). Види станів процесу визначаються конкретною ОС і системою керування процесами, але містять здебільшого такі стани:

Породження - створення процессу і підготовка до першого

виконання в системі;

Готовності - процес готовий до виконання і чекає

звільнення процесора;

Виконання - процес виконується на процесорі;

Блокування - процес призупинений через очікування

визначеної події: завершення введення даних, завершення заданого часу очікування, сигналу від іншого процесу, звільнення ресурсу та ін.;

Завершення - нормальне або аварійне завершення виконання процесу.

Причинами переходів є операції, які виконуються ОС над процесом: породження і завершення процесу, блокування і розблокування, завершення кванта часу роботи процесора, операції введення-виведення та ін.

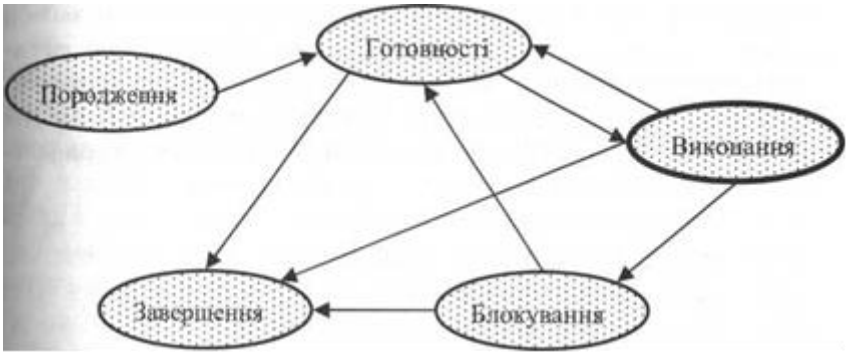


Рис. 2. Стани процесу

Процес також може мати особливий стан, який отримав назву ***тупик (deadlock).*** Процес у тупиковому стані блокований і очікує на подію, яка ніколи не відбудеться (сигналу від іншого процесу, звільнення ресурсу, введення даних та інше). Це зумовлює неможливість

його продовження і, як наслідок, - зависання програми в цілому. Тупики - одна з головних проблем, які виникають під час виконання паралельних програм.

Боротьба з тупиками зводиться до таких дій:

- відвернення тупиків;
- запобігання тупикам;
- визначення тупика;
- ліквідація тупика.

У цілому тупики є істотною загрозою і боротьбі з ними слід приділяти особливу увагу під час розроблення та налогодження паралельної програми. Приклади і методи запобігання тупикам будуть розглянуті в розділах, у яких вивчається взаємодії процесів.

Засоби роботи з процесами розподіляються на бібліотечні та мовні. Прикладами перших є засоби бібліотек WinAPI, PVM, MPI, OpenMP, Pthread, POSIX, які реалізовані у вигляді набору ресурсів (функцій, змінних, типів). Вони дозволяють роботу з процесами в будь-яких мовах програмування, зокрема і тих, які не мають вбудованих засобів роботи з процесами. Мови програмування, такі як C#, Java або Ада, безпосередньо мають засоби роботи з процесами, які вбудовані в мови, що забезпечує ефективне створення, виконання та організацію взаємодії процесів.

Незалежно від реалізації (бібліотечної або мовної) засоби роботи з процесами мають забезпечити розробнику паралельного додатка такі можливості:

- об’явити процес (групу процесів);
- встановити пріоритет процесу;
- запустити процес на виконання,
- призупинити процес на визначений час;
- блокувати та розблокувати процес;
- організувати взаємодію з іншими процесами (синхронізація або передавання даних);
- завершити процес.

Існує декілька шляхів до створення процесу:

- через використання *спеціальних модулів* (класів), які дозволяють описати процес (Java, Ada);
- через так звані *потокові функції,* коли дії майбутнього потоку описуються за допомогою функції, яка потім використовується для створення процесу (WinAPI, C#);
- через *створення копій* програми (MPI, PVM);
- через визначення в програмі ***паралельних ділянок,*** які можуть виконуватися паралельно (Occam, OpenMP).

В деяких мовах (бібліотеках) паралельного програмування використовуються комбінації наведених підходів. Так, в мові C# для створення процесу застосовують спеціальний модуль, дії якого визначаються через потокову функцію, а в бібліотеці OpenMP визначаються паралельні ділянки і процеси створюються шляхом копіювання окремих частин програми.

Далі повернемось назад і повторимо, що *Взаємодія поцесів* пов'язана з двома основними завданнями:

- комунікацією процесів;

- синхронізацією процесів.

Комунікація процесів передбачає обмін даними між процесами і при цьому інформація передається від одного процесу до іншого в будь-якому напрямку.

Синхронізація включає узгодження поведінки процесів і залежність виконання одного процесу від подій, які можуть бути в іншому процесі. Тому іноді використовується словосполучення «*синхронізація за подією*».

Механізм реалізації взаємодії процесів залежить від виду використовуваної комп'ютерної системи і в загальному випадку грунтується або на використанні моделі спільних змінних (системи зі спільною пам'яттю), або на використанні моделі посилання повідомлень (системи з локальною пам'яттю).

В системах зі спільною пам'яттю взаємодія процесів на фізичному (апаратному) рівні виконується за допомогою спільної пам'яті. На програмному рівні комунікація процесів і синхронізація процесів виконується через спільні змінні. Для передавання даних процес записує ці дані в спільні змінні, звідки їх зчитує інший процес. Синхронізація виконуєтья за допомогою відповідних спільних змінних, які змінюються процесом, в якому відбувається подія, і зчитуються (перевіряються) процесом, який чекає на подію.

Використання спільних змінних, які доступні будь-якому процесу (це обов'язково глобальні змінні), приводить до того, що в програмі може відбуватися одночасне звертання процесів до спільних змінних. Це може привести до конфлікту процесів, або некоректної роботи програми. Тому програмування з використанням моделі спільних змінних пов'язане з необхідністю вирішення специфічних проблем, які загалом зводяться до розв'язання двох основних завдань – *взаємного виключення і синхронізації процесів.*

Взаємодія процесів, яка ґрунтується на посиланні повідомлень

Загальна схема

Загальна схема передачі повідомлень між процесами грунтується на використанні операцій `Передати ()` і `Прийняти()` [5]. За допомогою операції `Передати ()` процес відправляє дані іншому процесу, який отримує ці дані за допомогою операції `Прийняти ( )`:

Процес 1	Процес 2
Передати(Дані);	Прийняти(Дані);

Існують різноманітні схеми операцій ***Передати ()*** і ***Прийняти ():*** іменовані, синхронні, асинхронні, з блокуванням або без блокування та інше.

Іменована схема потребує застосування імен взаємодіючих процесів в операціях ***Передати ()*** і ***Прийняти () :***

```
Передати (ім'я_Процесу_Отримувача, Дані)
Прийняти (ім'я_Процесу_Відправника, Дані)
```

При застосуванні непрямої схеми взємодії процесів з використанням буферу, каналу, лінку або труби (***pipe***) треба вказувати ім'я засобу, який застосовується:

Передати (Буфер, Дані)
Прийняти (Буфер, Дані)

Схема, в якій вказані імена обох процесів - відправника і отримувача, має назву ***симетричної. Асиметрична*** схема потребує вказати ім'я лише одного процесу, до якого звертаються. Асиметрична схема вигідна в моделі ***клієнт-сервер,*** коли процес-сервер не знає і не повинен знати, з яким процесом-клієнтом він буде взаємодіяти.

Існує кілька варіантів непрямої взаємодії процесів:

- один процес з одним процесом:
- один процес зкількома процесами;
- кілька процесів з одним процесом;
- кілька процесів з кількома процесами;

Передача повідомлень може бути також синхронною або асинхронною. Синхронна схема ґрунтується на тому, що процес, який виконує операцію ***Передати ()*** або ***Прийняти ()***, блокується, поки другий процес не вийде на приймання повідомлення (операція ***Прийнятни)*** або на передавання повідомлення (операція ***Передати ( )*.**

Оптимальною з погляду розробника є можливість використання різних видів операцій передавання-приймання повідомлень, як це зроблено в бібліотеці MPI, яка містить набір з кількох видів операцій ***Передати ()*** і ***Прийняти ()*** .

Далі коротко розглянемо бібліотеку MPI, так як MPI використовується при виконанні лабораторних робіт.

Бібліотека (інтерфейс) MPI (Message Passing Interface) визначає стандарт для програмування в комп'ютерних системах з розподіленою пам'яттю. MPI містить набір утиліт і бібліотечних функцій, що дозволяє проектувати паралельні додатки для мов, які не мають вбудованих засобів роботи з процесами.

Повідомлення. Процеси в MPI взаємодіють через посилання повідомлень. При цьому можливий обмін даними і синхронізація процесів. Розділяють процес-відправник і процес-отримувач.

У MPI повідомлення містить такі компоненти:

- блок даних (тип void*);
- тип даних (тип **`MPl_Datatype);`**
- кількість даних (тип **`int`** , кількість одиниць цього типу в блоці повідомлення.

Використовують таку інформацію про відправника і отримувача:

- комунікатор групи працюючих паралельних процесів (тип MPI_Comm);
- ранг відправника (номер процесу- отримувача в комунікаторі (тип int);
- ранг отримувача (номер процесу-відправника в комунікаторі, (тип int).

Крім того, для організації повідомлення використовують тег повідомлення - ціле число (тип int), що надається повідомленню відправником. Одержувач може вказати, що він буде приймати тільки ті повідомлення, які мають цей тег.

Обмін повідомлень має чотири форми:

1) попарний - повідомлення посилається одним процесом іншому одному процесу;

2) колективний - один процес передає повідомлення всім процесам його групи, які об'єднані за допомогою комунікатора;

3) асинхронний - процес-відправник не чекає, поки процес- отримувач отримує повідомлення, яке в цьому випадку копіюється в буфер;

4) синхронний - процес-відправник блокується, поки процес- отримувач не отримує повідомлення; процес- отримувач блокується, поки процес-відправник не буде готовий послати повідомлення.

Попарний обмін повідомленнями. Існує чотири види обміну для двох задач: синхронний, асинхронний, з блокуванням, без блокування. У MPI використовують чотири режими обміну (блокована і неблокована форми):

1) стандартне передавання; завершується одразу після відправлення повідомлення;

2) синхронне передавання; не завершується, поки не буде отримане повідомлення;

3) буферізоване передавання; завершується одразу же після копіювання повідомлення в системний буфер;

4) передавання по готовності; передавання розпочинається тільки тоді, коли адресат ініціює приймання повідомлення.

Домовленість про ім'я функцій обміну двох задач.

Для передавання:

MPI _ [I] [R,S,B] Send;

Префікс [I] означає неблокований режим; [R] - в міру готовності; [S] - синхронізований; [B] - буферизований.

Стандартний обмін не потребує жодного префікса. Таки чином, існує вісім різновидів передавання повідомлень.

Для приймання:

MPI_[I] Recv ;

Обмін із блокуванням процесів. Процес, який виконав посилання повідомлення блокується, поки інший процес не отримає це повідомлення. Процес, який приймає повідомлення, також буде блокований, поки він не отримує це повідомлення.

Надіслати повідомлення можна за допомогою функції MPI_Send ():

```
int MPI_Send(void *buffer, int count,
MPI_Datatype tp, int rg, int tag, MPI_Comm cm);
```

де:

buffer - адреса буфера з даними;

count - кількість елементів у буфері;

tp - тип даних для елементів у буфері;

rg - ранг отримувача в групі;

tag - тег повідомлення;

cm - комунікатор.

Отримати повідомлення можно за допомогою функції MPI_Recv():

```
int MPI_Recv(void *buffer, int count,
  MPI_Datatype tp, int rg, int tag, MPI_Comm cm,
  MPI_Status *st);
```

де:

buffer - адреса буфера, де слід розмістити отримане повідомлення;

count - максимальна кількість елементів у буфері;

tp - тип даних для елементів у буфері;

sr - ранг відправника в групі,

tag - тег повідомлення;

cm - комунікатор;

st - інформація про отримання повідомлення.

Обмін без блокування процесів. Операції передавання (приймання) повідомлень виконуються без блокування. Тобто передавальний процес відправляє повідомлення і не чекає на його отримання іншим процесом.

Для неблокованої взаємодії використовують відповідно функції MPI_ISendOi MPI_IReceive() .

Колективний обмін повідомленнями. Для реалізації колективного обміну застосовують поняття ***комунікатор*** - ідентифікатор групи процесів, які можуть обмінюватися повідомленнями. Ідентифікатор комунікатора використовують як аргумент у всіх функціях, які здійснюють обмін повідомленнями.

Комунікатор дозволяє організовувати колективні обміни всередині групи, ізолювати обміни різних груп процесів, ураховувати особливості структури розподіленої комп'ютерної системи.

Функції для роботи з комунікатором:

Кількість процесів в комунікаторі (групі):

int MPI_Comm_size(MPI_Comm cm, int *res);

де:

cm - комунікатор;

res - покажчик на результат (кількість процесів)

Номер (ранг) процесу в комунікаторі (групі):

int MPI_Comm_rank(MPI_Comm cm, int *rank);

де:

cm - комунікатор;

rank - покажчик результату (ранг).

Колективній обмін повідомленнями дозволяє оптимізувати організацію кількох попарних обмінів.

Для колективного розсилання даних з одного процесу в групі всім іншим процесам в групі використовують функцію MPI_Beast():

int MPI_Bcast(void *buffer, int count,

MPI_Datatype tp, int root, MPI_Comm cm)

де:

buffer - адреса буфера з відправленими даними для процесу з номером root; адрес буфера з даними, де слід розміщувати отримане повідомлення для інших процесів групи cm;

count - кількість елементів у буфері для процесу з номером root; максимальна кількість елементів у буфері з даними для інших процесів групи cm ;

tp - тип даних для елементів у буфері;

root - ранг відправника в групі cm;

cm - комунікатор.

Для колективного збору даних з кількох процесів групи в один процес групи використовують функцію MPI_Reduce () :

```
int MPIReduce (void *sbuffer, void *rbuffer,
                int count, MPI_Datatype tp,
                MPI_Op op, int root, MPI_Comm cm);
```

де:

sbuffer - адреса буфера;

rbuffer - покажчик буфера, де треба отримати результат; ви¬ користовується тільки для процесу з номером root у групі cm ;

count - кількість елементів у буфері;

tp - тип даних для елементів у буфері;

op - ідентифікатор операції (таб. 1);

root - ранг одержувача в групі cm ;

cm - комунікатор.

Таблиця 1. Деякі функції бібліотеки MPI

Функція	Операція
MPI_Send()	Переслати повідомлення. Блокуватися, поки воно не буде отримано
MPI_Recv()	Отримати повідомлення. Блокуватися, поки воно не буде надіслано
MPI_ISend()	Переслати повідомлення без блокування
MPI_IRecv()	Отримати повідомлення без блокування
MPI Probe()	З'ясувати, чи отримано повідомлення (блокуватися до отримання повідомлення)
MPI_IProbe()	З'ясувати, чи отримано повідомлення (без блокування)
MPI_Bcast()	Надіслати повідомлення всім процесам в групі
MPI_Reduce()	Отримати дані від усіх процесів у групі
MPI GatherO	Зібрати дані від всіх процесів
MPI_Scatter()	Розсилати дані усім процесам у групі. Дані - одного розміру.
MPI_Scatterv()	Розсилати дані всім процесам у групі. Дані можуть бути різного розміру.
MPI_Alltoall()	Розсилати дані всім процесам в групі зі всіх процесів групи. Дані одного розміру.
MPI AlltoallvO	Розсилати дані всім процесам у групі зі всіх процесів групи. Дані можуть бути різного розміру.

Операції, які можна виконувати в процесі-отримувачі над даними, які збираються функцією **MPi_** Reduce **(),** наприклад, формувати суму отриманих даних, наведено в табл. 2.

Таблиця 2. Операції в функції **MPI** ReduceQ

Код	Операція
MPI Max o	Максимум
MPI Min()	Мінімум
MPI SUMO	Сума
MPI PROD()	Додаток
MPI LAND()	Логічне «І»
MPI BANDO	Побітове «І»
MPI LOR()	Логічне «Або»
MPI BORO	Побітове «Або»
MPI LXOR()	Логічне «Виключне Або»
MPI BXOR()	Побітове «Виключне Або»
MPI MAXLOCO	Максимум та його позиція
MPI MINLOCO	Мінімум та його позиція

Якщо треба, щоб результат виконання функції MPI_Reduce() був отриманий не тільки одним процесом в групі (root), а всіма процесами у групі, то для цього використовують функцію MPI_Allreduce () :

```
int MPI_Allreduce(void *sbuffer, void *rbuffer,
    int count, MPI_Datatype tp, MPI_Op op,
    MPI_Comm cm);
```

де:

sbuffer - адреса буфера з данимі;

rbuffer - покажчик буфера, де треба отримати результат;

count - кількість елементів у буфері;

tp - тип даних для елементів у буфері;

op - ідентифікатор операції;

cm - комунікатор.

Результат формується в буфері в усіх процесорах групи cm.

Синхронізацію процесів в групі можно виконати за допомогою функції MPI_Barrier():

```
int MPI_Barrier(MPI_Comm cm);
```

де cm - комунікатор.

Ця функція блокує процес, який її викликав, доти, поки всі процеси групи з ідентифікатором *cm* не викличуть цю функцію.

В **моделі, яка базується на відправці повідомлень**, задачі працюють в окремих адресних просторах (не мають доступу до пам’яті сусідів) та виконують передачу даних та синхронізацію за допомогою відправки повідомлень. Повідомлення складається з ідентифікатора задачі-відправника, ідентифікатора задачі-отримувача та даних (рис.3). Ця модель найкраще підходить для паралельних обчислюальних систем з локальною пам’яттю. В паралельних обчислюальних систем зі спільною пам’яттю також можна застосувати цю модель: достатньо не використовувати спільних змінних в задачах.

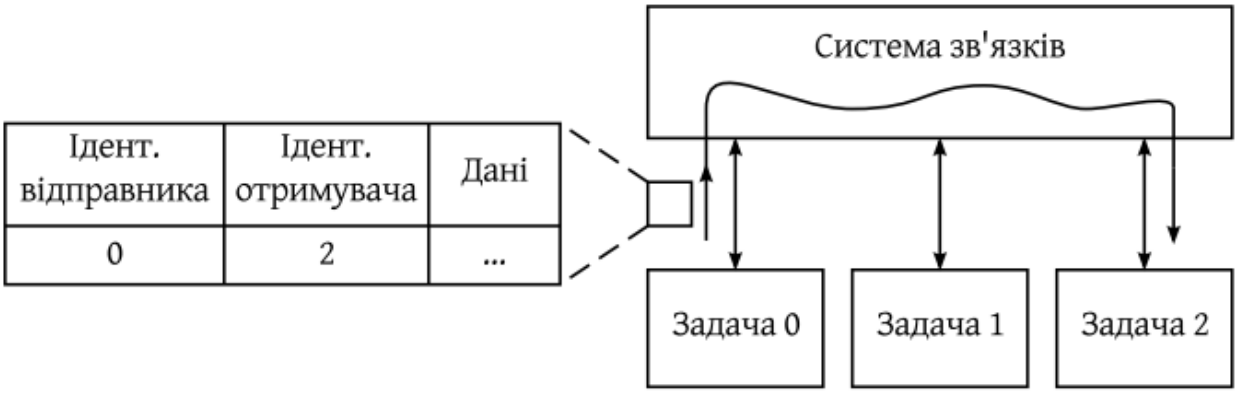


Рис. 3. Передача повідомлень

Перевагою даної моделі є те, що некоректний доступ до спільної пам’яті неможливий через те, що спільна пам’ять відсутня.

Недоліком є те, що програміст відповідає за передачу необхідних даних між задачами. Код, який забезпечує передачу даних між задачами, може становити велику частину всього коду.

Модель, яка базується на спільних змінних, розглядається на прикладі OpenMP. Компілятор GCC, який найчастіше встановлюється на кластерних системах під керуванням UNIX-подібних операційних систем, містить реалізацію OpenMP.

Для передачі даних між вузлами в кластерних системах найчастіше застосовується інтерфейс MPI, який відповідає моделі, яка базується на відправці повідомлень.

Література

4. Дейтел Д. Введение в операціонные системы. – М. Мир, 1989. – 360 с.

5. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. Учебное пособие – М.: Издательство Московского университета, 2011. – 563с .

6. Воеводин В. В. Параллельные вычисления. - СПб. : БХВ- Петербург, 2002. — 608 с.

4. С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко Засоби паралельного програмування – К., 2011. – 154 с.

5. І.А. Жуков, О.В. Корочкін. Паралельні та розподілені обчислення – Київ: «Корнійчук», 2014. – 284 с.

3 *Обчислювальний Кластер.*

3.1. Обчислювальний Кластер. Основи побудови

Кластери – багатопроцесорні ЕОМ, побудовані із елементів серійного виробництва. Це відноситься не тільки до апаратури, але і до програмного забезпечення. Основною операційною системою для кластерів являється безплатна ОС Linux.

В свою чергу кластери можна розділити на дві помітно різні по продуктивності гілки:

• кластери типу Beowulf, які будуються на базі звичайної локальної мережі ПЕОМ. Щоб отримати такий кластер, до мережі ПЕОМ потрібно добавити тільки системне програмне забезпечення, яке безплатно розповсюджується в Інтернет;

• монолітні кластери, все обладнання яких компактно розміщене в спеціалізованих шафах. Це дуже швидкі машини, кількість процесорів в яких може досягати сотень і тисяч. Процесори в монолітних кластерах не можуть використовуватися в персональном режимі.

Можна сказати, що кластер – це група обчислювальних вузлів, об’єднаних швидкісною мережею, яка являє собою для користувача єдиний апаратний ресурс, а кластерні обчислення – це особлива технологія високопродуктивних обчислень, яка зародилася разом з розвитком комунікаційних засобів і стала прекрасною альтернативою використанню надпотужних комп’ютерів. Коли з’явилися доступні канали зв’язку з високою пропускною здатністю, з’явилась концепція віртуального супернкомп’ютера, де масштабна задача виконується сумісно в єдиній мережі кластером звичайних комп’ютерів. До розряду кластерних обчислень можна віднести всякі паралельні обчислення, де всі комп’ютери системи використовуються як один уніфікований ресурс. Існує кілька типів кластерів: кластери високої доступності, MPP-кластери, високопродуктивні кластери, кластери розподіленої загрузки.

Обчислювальний кластер.

Це система, зазвичай, складається з одного серверного вузла і одного або більше клієнтських вузлів, з'єднаних за допомогою Ethernet, або деякої іншої мережі. Обчислювальний кластер, як правило, працює під управлінням однієї з версій ОС Linux – багатокористувацької операційної системи і обчислювальний кластер – це технологія кластерізації комп’ютерів, які працюють під управлінням ОС Linux на різновид паралельного віртуального суперкомп’ютеру. Ключем до поняття кластерних обчислень є ідея єдиного образу системи. У кластерах передбачена плавна зміна розмірів і подолання відмов: коли один підвузол у кластері відмовляє, то інший готовий зайняти його місце так, щоб додатки (включаючи і його користувачів) практично цього не помітили.

Кластер складається з окремих машин (вузлів) і об'єднуючої їх мережі (комутатора). Крім ОС, необхідно встановити та налаштувати мережеві драйвери, компілятори, програмне забезпечення для підтримки паралельного програмування і розподілу обчислювального навантаження.

Вузли кластера: підходящим вибором в даний момент є системи на базі процесорів Intel (наприклад, існуючий кластер НТУУ «КПІ» - один із найпотужніших в Україні - містить 44 вузли з двома чотириядерними процесорами Intel Xeon і 68 вузлів з двома двоядерними процесорами Intel Xeon, тобто всього 224 процесори і 624 ядра) . Варто встановити на кожен вузол не менше 1Gb оперативної пам'яті. Бажано 2-4Gb. Одну з машин слід виділити в якості центральної (консоль кластеру), куди можна (але не обов'язково) встановити досить великий жорсткий диск, можливо більш потужний процесор і більше пам'яті, ніж на інші (робочі) вузли. Робити консоль кластеру більш потужною машиною має сенс, якщо необхідно мати на цьому комп'ютері крім інтерфейсу командного рядка більш зручне операційне оточення, наприклад віконний менеджер (KDE, Gnome), офісні програми, програми візуалізації даних і т.п.

Будова кластеру

Розгортання кластеру - завдання не надто складне. Для цього підійде будь-який дистрибутив. Який саме з дистрибутивів Linux ставити в якості базової ОС - не має значення. Ubuntu, Mandriva, Alt Linux, Red Hat, SuSE. Вибір залежить тільки від уподобань користувача.

Отже, щоб розвернути кластер, використовуючи дистрибутив загального призначення, слід виконувати наступне:

1. Встановити операційну систему на комп'ютер, який буде виступати в ролі консолі кластеру. Тобто на цьому комп'ютері будуть компілюватися і запускатися паралельні програми. Іншими словами, за цим комп'ютером буде сидіти людина, запускати програми і дивитися, що вийшло.
2. Після інсталяції базової ОС на консолі кластеру (якщо це не зроблено в процесі передньої установки) необхідно встановити необхідні компілятори (фортран, С) і всі необхідні бібліотеки, desktop environment (GNOME або KDE), текстові редактори і т. д., тобто перетворити цей комп'ютер в робочу станцію розробника.
3. Встановити з репозитарію або з вихідного пакет MPICH, який являється спеціальною реалізацією MPI для Globus Toolkit.
4. Описати в /etc/hosts майбутні вузли кластеру, в тім рахунку консоль кластеру.
5. Встановити NFS і розшарувати для всіх вузлів кластеру якусь директорію, в якій будуть розміщуватися виконавчі модулі паралельних програм та файли даних, якими ці програми будуть користуватися в процесі своєї роботи.
6. Встановити на консолі кластеру ssh-клієнт (обов'язково) та ssh-сервер (опціоно, якщо буде надаватися доступ до консолі кластеру по мережі).
7. На всіх вузлах кластеру встановити операційну систему, бібліотеки, необхідні для виконання користувацьких паралельних програм, встановити MPICH, NFS-client, ssh-server. Якщо вузли кластеру в цілях економії ресурсів

навантажуються в runlevel 3, то ставити GNOME або KDE не треба. Максимум - поставити ряд бібліотек, якщо вони потрібні для користувача.
8. Описати в /etc/hosts всіх вузлів кластеру майбутні вузли вашого кластеру, в тім рахунку і консоль кластеру.
9. На всіх вузлах кластеру необхідно автоматично при завантаженні монтувати розшарений ресурс. Причому, шлях до цього ресурсу має бути однаковий, як на консолі кластеру, так і на його вузлах. Наприклад, якщо на консолі кластеру дати доступ каталогу /home/mpiuser/data, то на вузлах кластеру цей ресурс також має бути змонтований в /home/mpiuser/data.
10. На всіх вузлах кластеру забезпечити безпарольний доступу по ssh для консолі кластеру.
Оскільки від мережі залежить ефективність роботи кластеру, то бажано зробити наступне. Необхідно, щоб функціонування мережевої файлової системи NFS не заважало обміну даними, який здійснюють між собою частини паралельної програми, які працюють в різних вузлах. Щоб це здійснити, необхідно фізично розділити мережу на два сегменти. В одному сегменті буде працювати NFS, в іншому - відбуватиметься обмін даними між частинами програми.

Організація мережі обчислювального кластеру

Мережа - це модульна і адаптуюча комутаційна система, яку можна налаштувати відповідно до самих різних вимог. Її модульність полегшує додавання нових компонентів, або переміщення існуючих, а адаптивність спрощує внесення змін і удосконалень.

Мережа кластеру нічим принципово не відрізняється від мережі робочих станцій, тому в найпростішому випадку для побудови кластеру необхідні звичайні мережеві карти та хаби / комутатори, які використовувалися б при облаштуванні якогось комп'ютерного класу. Однак, у випадку кластеру є одна особливість. Мережа кластеру в першу чергу призначена не для зв'язку машин, а для зв'язку обчислювальних процесів. Тому чим вищою буде пропускна здатність мережі, тим швидше будуть виконуватись паралельні завдання, запущені на кластері, отже робочі характеристики мережі набувають першочергового значення.

Для побудови обчислювальних кластерів використовують найрізноманітніше мережеве обладнання. Але слід пам’ятати, що пропускна здатність мережі, що зв'язує вузли кластеру, у багатьох випадках виявляється вирішальною для продуктивності кластеру.

Використовуване мережеве обладнання характеризують зазвичай двома параметрами:

Латентність - це середній час між викликом функції передачі даних і самою передачею. Час витрачається на адресацію інформації, спрацювання проміжних мережних пристроїв, інші накладні витрати, що виникають при передачі даних.

Фактично пропускна здатність і латентність не тільки характеризують кластер, але й обмежують клас задач, які можуть ефективно вирішуватися на ньому. Так, якщо завдання вимагає частої передачі даних, кластер, який

використовує мережеве обладнання з великою латентністю (наприклад, GigabitEthernet), буде велику частину часу витрачати навіть не на передачу даних між процесами, а на встановлення зв'язку, в той час як вузли будуть простоювати, і ми не отримаємо значного збільшення продуктивності. Втім, якщо пересилаються великі обсяги даних, вплив періоду латентності на ефективність кластеру може знижуватися за рахунок того, що сама передача вимагатиме досить великого часу, може бути навіть у рази більшою за період латентності.

Мережеві карти

В якості мережевих адаптерів можна використовувати будь-які доступні карти, які підтримують роботу в стандартах 100BaseTx і GigabitEthernet.

Комутатори

Іншим важливим елементом мережі кластеру є пристрої комутації мережевих каналів. Комутатори та інші елементи мережевої структури використовуються для забезпечення комунікацій між процесорами, для підтримки паралельного програмування і різних функцій керування. Для паралельного програмування організації взаємодії між процесами (Inter Process Communication, IPC) широко використовується комутатор Myrinet-2000 - швидкий, добре масштабований широкосмуговий пристрій.

Паралельна віртуальна машина (PVM)

Основою обчислювального середовища кластеру є паралельна віртуальна машина (PVM). PVM - це пакет програм, який дозволяє використовувати пов'язаний в локальну мережу набір різнорідних комп'ютерів, як один великий паралельний комп'ютер. Таким чином, проблема великих обчислень може бути досить ефективно вирішена за рахунок використання сукупної потужності та пам'яті великої кількості комп'ютерів. Пакет програм PVM легко переноситься на будь-яку платформу. Вихідні тексти, вільно розповсюджувані netlib, були скомпільовані на комп'ютерах, починаючи від laptop і до CRAY.

У загальному випадку кількість завдань може перевершувати число процесорів, включених в PVM. Крім того, до складу PVM можна включати досить різнорідні обчислювальні машини, несумісні з систем команд і форматів даних. Отже, паралельною віртуальною машиною може стати як окремо взятий ПК, так і локальна мережа, що включає в себе суперкомп'ютери з паралельною архітектурою, універсальні ЕОМ, графічні робочі станції і все ті ж малопотужні ПК. Важливо лише, щоб процеси, які включаються до PVM-обчислювальних засобів були інформацією у використовуваному програмному забезпеченні PVM. Завдяки цьому користувач може вважати, що він спілкується з однією обчислювальною машиною, в якій можливе паралельне виконання безлічі завдань.

PVM дозволяє користувачам використовувати існуючі апаратні засоби, для вирішення набагато більш складних завдань при мінімальній додаткової вартості. Головна мета використання PVM – це підвищення швидкості

обчислень за рахунок їх паралельного виконання. Функціонування PVM засноване на механізмах обміну інформацією між завданнями, виконуваними в її середовищі. У цьому відношенні найбільш зручно реалізовувати PVM в рамках багатопроцесорних обчислювальних комплексів, виділивши віртуальній машині кілька процесорів і загальну або індивідуальну (залежно від умов) оперативну пам’ять. Використання PVM доспускається як на багатопроцесорних комп'ютерах (SMP), так і на обчислювальних комплексах, побудованих за допомогою кластерної технології. При використанні PVM, як правило, значно спрощуються проблеми швидкого інформаційного обміну між завданнями, а також проблеми узгодження форматів представлення даних між завданнями, виконуваними на різних процесорах.

Ефективне програмування для PVM починається з того, що алгоритм обчислень слід адаптувати до складу PVM і до її характеристик. Це дуже творче завдання, яке в багатьох випадках повинне вирішуватися програмістом. Крім завдання розпаралелювання обчислень з необхідністю виникає і завдання керування обчислювальним процесом, координації дій і завдань учасників цього процесу. Іноді для керування потрібно створювати спеціальну задачу, яка сама не беручи участь в обчисленнях, забезпечує узгоджену роботу решти завдань.

При паралельних обчисленнях необхідно програмувати спеціальні дії з координації роботи завдань, такі як процеси запуску задач на процесорах кластеру, управління обміном даних між завданнями та ін. Найбільш простий спосіб організації паралельного процесу виглядає наступним чином. Спочатку запускається одне завдання (master), яке в колективі завдань буде відображати функції координатора робіт. Це завдання виробляє деякі підготовчі дії, наприклад, ініціалізація початкових умов, після чого запускає інші завдання (slaves), яким може відповідати або той же виконуваний файл, або різні виконувані файли. Такий варіант організації паралельних обчислень переважно використовується при ускладненні логіки керування обчислювальним процесом, а також коли алгоритми, реалізовані в різних завданнях, істотно розрізняються, або є великий обсяг операцій (наприклад, введення - виведення), які обслуговують обчислювальний процес в цілому.

Отже резюмуючи розглянуті питання, кластери – це багатопроцесорні ЕОМ, побудовані на елементах серійного виробництва. Це відноситься не тільки до апаратури, але і до програмного забезпечення. Основою операціної системои для кластерів являеться ОС Linux. В свою чергу кластери можна розділити на два помітно різні по спроможності класи:

• кластери типу Beowulf, які будуються на базі звичайної локальнї мережі ПЕОМ. Щоб отримати такий кластер, до мережі ПЕОМ потрібно добавити тільки системне програмне забезпечення, яке безоплатно розповсюджується в інтернеті;

• монолітні кластери, все обладнання яких компактно розміщене в спеціалізованих шкафах. Це дуже швидкі машини, кількість процесорів в яких може досягати сотень та тисяч. Процесори в монолітних кластерах не можуть викороистовуватися в персональному режимі.

Проект Beowulf розпочався в 1994р в центрі NASA: був зібраний 16-процесорний кластер на процесорах Intel 486DX4/100 Мгц. На кожному вузлі було встановлено по 16 Мбайт ОЗП і по 3 мережевих карт для звичайної мережі Ethernet. Були розроблені спеціальні драйвери для роботи в такій конфігурації, які розподіляли трафік між доступними мережевими картами. Результат експлуатації кластеру виявився вдалим і тепер переважна більшість суперкомп'ютерних систем по списку Top500 відносяться до кластерних систем. Обчичлювальний кластер – це мультикомп'ютер, який складається з певної кількості окремих комп'ютерів (вузлів), зв'язаних між собою єдиною комунікаційною системою: кожний вузол має свою локальну оперативну пам'ять, загальної фізичної оперативної пам'яті для вузлів не існує. Вузли кластера – це повноцінні комп'ютери. Якщо вузли кластеру побудовані на мультипроцесорах (мультипроцесорні комп'ютери з загальною пам'яттю), то такий кластер називається SMP-кластером. Якщо всі вузли кластеру мають однакову архітектуру і продуктивність, то такий кластер є однорідним, ящо ні – то неоднорідним. Перші Beowulf-кластери використовували 10-мегабітну мережу Ethernet. Сьогодні використовується мережа Gigabit Ethernet на базі комутаторів. Існують і інші комунікаційні технології.
Найпоширинішим є використання однорідних кластерів. Один сервер виконує функції керуючого вузла кластеру. На цьому комп'ютері встановлене програмне забезпечення, яке активує обчислювальні вузли при завантаженні операційної системи і керує запуском програм на кластері. Завдання користувачів виконуються на обчислювальних вузлах, де вони розподіляються таким чином, що на кожний процесор розподіляється не більше одного обчислювального процесу. Користувачі мають термінальний доступ до головної машини кластеру (цей сервер забезпечує зв'язок кластеру із завнішньою мережею через корпоративну локальну мережу, або через Інтернет). Найчастіше для термінального доступу використовується протокол SSH. На цьому ж сервері розміщені домашні каталоги користувачів. У деяких конфігураціях для доступу користувачів і розміщення домашніх каталогів виділяється окремий сервер – так званий сервер доступу. Доступ на вузли кластеру або закритий для користувачів, або можливий, наприклад, для ручного управління компіляцією завдання.

Програми на кластері виконуються в пакетному режимі: користувач не має безпосередньої взаємодії з програмою, прорама не може очікувати введення даних із клавіатури й виводити їх безпосередньо на екран. Більше того, програма користувача може працювати тоді, коли користувач не підключений до кластеру.

Як вже вказувалося Обчислювальний кластер, як правило, працює під управлінням однієї з версій ОС Linux – багатокористувацької ОС. Всі вузли кластеру мають доступ до загальної файлової системи, яка розміщується на файл-сервері. Файл може бути створений, наприклад, на головній машині, або на обчислювальному вузлі, а потім прочитаний під тим же ім'ям на іншому вузлі. Запис в один файл одночасно з різних вузлів неможливий, але запис в

різні файли припустимий. Крім загальної файлової системи можуть використовуватися локальні диски на обчислювальних вузлах кластеру. Їх використовують програми для зберігання тимчасових файлів. Після закінчення роботи програми ці файли видаляються.

Задіяти обчислювальні потужності кластеру можна різними способами :
- виконувати певну кількість однопроцесорних задач, якщо потрібно провести певну кількість незалежних обчислювальних експериментів з різними вхідними даними, а всі дані розміщуються в оперативній пам'яті, доступній одному процесу;

- використовувати готові пакети програм, в яких реалізовані паралельні обчислення. Для деяких задач доступні безкоштовні паралельні програми, які можна використовувати на кластері;

- здійснювати у власних програмах виклик додатків з стандартних бібліотек для паралельних обчислень;

- створювати власні програми.

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення обчислювального кластеру

2. Перерахуйте особливості програмування обчислювального кластеру

3. Що таке координація розрізнених ресурсів?

4. Охарактеризуйте Обчислювальний комп'ютерний кластер?

5. Наведіть будову КК

6. Охарактеризуйте мережу кластеру

7. Як можна створити кафедральний кластер?

8. Що таке Латентність?

9. Що таке PVM?

10. Охарактеризуйте типові вузли кластеру

11.Що Ви знаєте про кластер НТУУ «КПІ ім. Ігоря Сікорського»?

Література

1. Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко– К., 2011. – 154 с.
2. Жуков І.А. Паралельні та розподілені обчислення [Текст] / І.А. Жуков, О.В. Корочкін. – Київ: «Корнійчук», 2014. – 284 с.
3. Г.И.Шпаковский, В.И.Стецюренко, А.Е. Верхотуров, Н.В. Серикова. Применение

технологии MPI в Грид. – Минск, БГУ, 2008 , - 137 с.

4. Інструкція по використанню кластеру НТУУ «КПІ ім. Ігоря Сікорського» [Електронний ресурс]. – Режим доступу: http://hpcc.kpi.ua/uk/

3. 2. *Створення кафедрального обчислюваль кластеру*

Обчислювальний кластер - це мультикомп'ютерна архітектура, яка переважно використовується для паралельних обчислень. Однако, справжній класичний кластер це далеко не просте створіння і його побудова під силу тільки дуже потужним структурам.

Створення навчальної кафедральної системи для освоєння паралельних обчислень в комп'ютерній мережі є практичною реалізацією методів розпаралелювальня обчислювальних процесів.

У сучасному світі для освоєння паралельних обчислень широко використовуються кластерні системи, які стали робочим інструментом в наукових дослідженнях та інженерних розрахунках. Їх по праву називають основною технологічною зброєю XXI ст. Неохідність в таких обчисленнях обумовлена ускладненням математичних моделей, коли моделюється не один об'єкт, а велика їх кількість.

Існуючий кластер НТУУ «КПІ ім. Ігоря Сікорського» є одним із найпотужніших в Україні (містить 44 вузли з двома чотириядерними процесорами Intel Xeon і 68 вузлів з двома двоядерними процесорами Intel Xeon, тобто всього 224 процесори і 624 ядра) і не взмозі забезпечити потреби повсякденного широкого використання всього існуючого студентського загалу НТУУ. Цей кластер можна розглядати як серйозний інструмент для фахівців з відповідним практичним досвідом, які набили свої «гулі» на простіших інструментах. Мабуть саме тому студенти ІПСА вчаться робити обчислення на кластері інституту теоретичної фізики НАНУ. Пару років назад я пробував робили те саме, але це не прижилося в нас. Створення навчальної кафедральної системи для освоєння паралельних обчислень в комп'ютерній мережі для потреб окремої кафедри чи факультету для широкого залучення їх студентського загалу є актуальною задачею.

Як створити навчальний кафедральний класте

На ФПМ є 12 комп'ютерних класів із загальною кількістю комп'ютерів 119. На кафедрі програмного забезпечення комп'ютерних систем факультету прикладної математики є три комп'ютерних класи, які налічують 28 сучасних комп'ютерів, об'єднаних в локальну мережу, є вихід до Internet. На їх основі і можна побудувати обчислювальну навчальну кафедральну систему для освоєння паралельних обчислень для використання в навчальному процесі. Така обчислювальна система буде повільною із-за затримок в мережі, її

ефективність може поступатися реалізації паралельних обчислень навіть в одному хорошому комп'ютері, але для навчальних цілей це не так важливо. Важливішим є можливість здобуття студентами відповідного практичного досвіду без зайвого остраху зробити щось не так.
В свій час я пропогував створення такої системи силами магістрів професійних, але це не знайшло потрібної підтримки із-за необхідності виділення окремого спеціаліста на кафедрі для обслуговування створеного кластеру.

Обчислювальний кластер - це не специфічний пакет програм, нова топологія мережі, або новітня модифікація ядра ОС, а технологія кластеризації комп'ютерів, які працюють під управлінням ОС Linux на різновид паралельного віртуального суперкомп'ютера. **Кластер складається з окремих машин (вузлів) і об'єднуючої їх мережі**. Крім ОС, необхідно встановити та налаштувати мережеві драйвери, компілятори, програмне забезпечення для підтримки паралельного програмування і розподілу обчислювального навантаження.

В загальному розгортання кластеру як такого - завдання актуальне та доволі не складне. Причому, для цього підійде будь-який дистрибутив. Який саме з дистрибутивів Linux ставити в якості базової ОС - не має значення. Ubuntu, Mandriva, Alt Linux, Red Hat, SuSE. Вибір залежить тільки від уподобань користувача.

Щоб розвернути кластер, використовуючи дистрибутив загального призначення, слід виконати наступне:
1. Встановити операційну систему на комп'ютер, який буде виступати в ролі консолі кластеру. Тобто на цьому комп'ютері будуть компілюватися і запускатися паралельні програми. Іншими словами, за цим комп'ютером буде сидіти людина, запускати програми і дивитися, що вийшло.
2. Після інсталяції базової ОС на консолі кластеру, якщо це не зроблено в процесі первісної установки, необхідно буде встановити необхідні компілятори (фортран, С) і всі необхідні бібліотеки, desktop environment (GNOME або KDE), текстові редактори і т. д., тобто перетворити цей комп'ютер в робочу станцію розробника.
3. Встановити з репозитарію або з вихідного пакет MPICH .
4. Описати в /etc/hosts майбутні вузли вашого кластеру, в тім рахунку і консоль кластеру.
5. Встановити NFS і розшарити для всіх вузлів кластера якусь директорію, в якій будуть розміщуватися виконавчі модулі паралельних програм та файли даних, якими ці програми будуть користуватися в процесі своєї роботи.
6. Встановити на консолі кластеру ssh-клієнт та ssh-сервер (опціонально, якщо буде надаватися доступ до консолі кластеру по мережі).
7. На всіх вузлах кластеру встановити операційну систему, бібліотеки,

необхідні для виконання користувальницьких паралельних програм, встановити MPICH, NFS-client, ssh-server. Поставити ряд бібліотек, які потрібні для користувача.
8. Описати в /etc/hosts всіх вузлів кластеру майбутні вузли вашого кластеру, в тім рахунку і консоль кластеру.
9. На всіх вузлах кластеру необхідно автоматично при завантаженні монтувати розшарений ресурс. Причому, шлях до цього ресурсу повинен бути однаковий, як на консолі кластеру, так і на його вузлах. Наприклад, якщо на консолі кластеру дати доступ каталогу /home/mpiuser/data, то на вузлах кластеру цей ресурс також має бути змонтований в /home/mpiuser/data.
10. На всіх вузлах кластеру забезпечити безпарольний доступу по ssh для консолі кластеру.

Оскільки від мережі прямо залежить ефективність роботи кластеру, то зробити наступне. Необхідно, щоб функціонування мережевої файлової системи NFS не заважало обміну даними, який здійснюють між собою частини паралельної програми, що працюють у різних вузлах. Щоб це здійснити, необхідно фізично розділити мережу на два сегменти. В одному сегменті буде працювати NFS, в іншому - відбуватиметься обмін даними між частинами програми.

Основою обчислювального середовища кластеру є паралельна віртуальна машина PVM. PVM - це пакет програм, який дозволяє використовувати пов'язаний в локальну мережу набір різнорідних комп'ютерів, які працюють під операційною системою Unix, як один великий паралельний комп'ютер. Таким чином, проблема великих обчислень може бути досить ефективно вирішена за рахунок використання сукупної потужності та пам'яті великої кількості комп'ютерів. Пакет програм PVM легко переноситься на будь-яку платформу. Вихідні тексти, вільно розповсюджувані netlib, були скомпільовані на комп'ютерах починаючи від laptop і до CRAY.

У загальному випадку кількість завдань може перевершувати кількість процесорів, включених в PVM. Крім того, до складу PVM можна включати досить різнорідні обчислювальні машини, несумісні з систем команд і форматів даних. Взагалі, паралельною віртуальною машиною може стати як окремо взятий ПК, так і локальна мережа, яка включає в себе суперкомп'ютери з паралельною архітектурою, універсальні ЕОМ, графічні робочі станції і все ті ж малопотужні ПК. Важливо лише, щоб процеси, які включаються до PVM обчислювальних засобах були інформацією у використовуваному програмному забезпеченні PVM. Завдяки цьому програмному забезпеченні користувач може вважати, що він спілкується з однією обчислювальною машиною, в якій можливе паралельне виконання безлічі завдань.

PVM дозволяє користувачам використовувати існуючі апаратні засоби, для вирішення набагато більш складних завдань при мінімальній додаткової

вартості. Сотні дослідних груп в усьому світі використовують PVM, щоб вирішити важливі наукові, технічні, і медичні проблеми, а так само використовують PVM як освітній інструмент, для викладання паралельного програмування. В даний час, PVM став де факто стандартом для розподілених обчислень.

Головна мета використання PVM – це підвищення швидкості обчислень за рахунок їх паралельного виконання. Функціонування PVM засноване на механізмах обміну інформацією між завданнями, виконуваними в її середовищі. У цьому відношенні найбільш зручно реалізовувати PVM в рамках багатопроцесорних обчислювальних комплексах, виділивши віртуальній машині кілька процесорів і загальну або індивідуальну (залежно від умов) оперативну пам’ять. Використання PVM доспускається як на багатопроцесорних комп'ютерах (SMP) так і на обчислювальних комплексах, побудованих за допомогою кластерної технології. При використанні PVM, як правило, значно спрощуються проблеми швидкого інформаційного обміну між завданнями, а також проблеми узгодження форматів представлення даних між завданнями, виконуваними на різних процесорах.

Ефективне програмування для PVM починається з того, що алгоритм обчислень слід адаптувати до складу PVM і до її характеристик. Це дуже творче завдання, яке в багатьох випадках повинне вирішуватися програмістом. Крім завдання розпаралелювання обчислень з необхідністю виникає і завдання керування обчислювальним процесом, координації дій учасників цього процесу. Іноді для керування доводиться створювати спеціальну задачу, яка сама не беручи участь в обчисленнях, забезпечує узгоджену роботу решти завдань .

Найбільш простий і популярний спосіб організації паралельного процесу виглядає наступним чином. Спочатку запускається одне завдання (master), яке в колективі завдань буде відображати функції координатора робіт. Це завдання виробляє деякі підготовчі дії, наприклад ініціалізація початкових умов, після чого запускає інші завдання (slaves), яким може відповідати або той же виконуваний файл, або різні виконувані файли. Такий варіант організації паралельних обчислень переважно використовується при ускладненні логіки керування обчислювальним процесом, а також коли алгоритми, реалізовані в різних завданнях, істотно розрізняються або є великий обсяг операцій (наприклад, введення - виведення), які обслуговують обчислювальний процес в цілому.

Практичне створення навчальної кафедральної системи.

Таку навчальну систему можно створити в невеликій організації чи вузівській кафедрі. Цей варіант позволяє навчитися: створювати мережу ,

інсталювати системне ПО (складна робота) та виконувати паралельні обчислення.

Для того, щоб побудувати паралельну систему на базі комп'ютерної мережі, потрібно:

1. Побудувати мережу комп'ютерів (в нас вона вже є).
2. Інсталювати системне програмне забезпечення.
3. Створити паралельну програму, виконати налаштування системи і програми.
4. Провести обчислення та оцінити їх ефективність.

Нижче приведений процес інсталяції мобільного варіанту з розрахунку, що комп'ютерна мережа з апаратурою та програмним забезпеченням уже створена:

1.На сайті www.globus.org по адресу http://www.globus.org/toolkit/downloads/4.0/ представлений перелік і адреси операційних систем та інших програмних комплексів, які можна використовувати для інсталяції одного із варіантів системи Грід. (у нас теж дуже спрощений варіант Грід)

2. По адресу http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html знаходиться опис порядку і всіх деталей інсталяції по варіанту quickstart (швидкий старт).

Для виконання цього пункту частину програм можна встановити попередньо (пункт 1), а частину – в процесі виконання (цього) пункту 2. Але при вконанні (цього) пункту 2 всякий раз потрібно переконуватись чи встановлене необхідне програмне забезпечення.

3. Після завершення інсталяції необхідно поверх нього встановити пакет MPICH-G2, який являється спеціальню реалізацією MPI для Globus Toolkit. Таким засобом являється пакет MPICH-G2 (www.globus.org/grid_software/computation/mpich-g2.php) - це спеціальна реалізація MPI для Globus Toolkit. 4. Накінець, потрібно запустити зі встановленим MPICH-G2 тест для вимірювання продуктивності паралельних систем HPL, призначених для рішення систем лінійних алгебраїчних рівнянь, який є развитком тесту LINPACK. Інформація по тесту HPL разміщена по адресу http://www.netlib.org/benchmark/hpl/index.html .

Для створення навчальої кафедральної системуи для практичного освоєння паралельних обчислень в комп'ютерній мережі потрібно використати кафедральні ресурси ауд.5/14 та 76/14. (необхідну для цього інформацію щодо консолі отримати у доцента Оная Миколи Володимировича).

MHICH (MPI Chameleon) – вільно розповсюджуєма реалізація MPI. Пакет доступний в вихідних кодах, підтримується в UNIX, Mac, Windows.

Література

1. Шпаковский В.И. др. Применение технологии MPI в Грид. Минск, 2008, 137с., стор. 129-136.
2. Г.И. Радченко. РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ. Учебное пособие. Челябинск, 2012,

4 *Суперкомп'ютер НТУУ «КПІ ім. Ігоря Сікорського».*

4. 1. Робота на обчислювальному кластері НТУУ «КПІ ім. Ігоря Сікорського»

Існуючий кластер НТУУ «КПІ» - один із найпотужніших в Україні - містить 44 вузли з двома чотириядерними процесорами Intel Xeon і 68 вузлів з двома двоядерними процесорами Intel Xeon, тобто всього 224 процесори і 624 ядра. Багато це, чи мало? На це можу сказати тільки те, що вже в 2009р суперкомп'ютер Jaguar - Cray XT5 включав в себе 224 162 ядра. Навожу це без сарказму, а з біллю в душі, бо на першій лекції я Вам розповідав, що справжній бум обчислювальної техніки розпочався в 1964р зі створення системи IBM-360 в США і що в той же час у нас в Україні і конкретно в Києві була створена наша ЕОМ Дніпро-2 аналогічного класу, яка нічим не поступалася американській (64-розрядна машина з 128-розрядною "плаваючою" арифметикою і аналогічною оперативною і зовнішньою пам'яттю). Я розповідав Вам, як героїчно ми її (Дніпро-2) знищили і закопали, після чого кинулися створювати український клон їхньої IBM-360. Самі ми до цього би не додумалися. І аналогічні "подвиги" ми продовжуємо здійснювати до сьогоднішнього дня.

Кластерна обчислювальна система Центру суперкомп'ютерних обчислень НТУУ «КПІ ім. Ігоря Сікорського» працює під управлінням операційної системи *Linux*. В *Linux* стандартним засобом віддаленого доступу є протокол *Secure Shell* (скорочено *SSH*). Віддалений доступ можливий як через Інтернет, так і з мережі НТУУ «КПІ ім. Ігоря Сікорського». Найбільш

популярні програми-клієнти, які дозволяють встановлювати зв'язок за протоколом *SSH* - це утиліти *ssh* в *Linux* та *PuTTY* для *Windows*.

Розглянемо роботу з програмою *PuTTY*, яку можна завантажити з сайту: http://www.putty.org/. Для запуску задач на кластерній системі необхідно мати на ній акаунт. Акаунт створюється адміністратором по заявці. Вам мають бути повідомлені наступні дані:

– логін вашого акаунта (далі *user*);
– пароль вашого акаунта (далі *pass*);
– IP-адресу кластеру або його символьне ім'я (далі *host*);
– порт, на який встановлюється *SSH*-з'єднання (далі *port*).

1. Віддалений доступ до командного рядка

1.1 Віконний варіант *PuTTY*

Після інсталяції програми *PuTTY* можна користуватись як віконним варіантом програми, так і консольним. Обидва варіанти представлені виконуваним файлом *putty.exe*. Запуск цього файлу без параметрів відкриває головне вікно програми: *Пуск* → *Програми* → *PuTTY* → *PuTTY*).

Віконна версія програми має перевагу в тому, що вона дозволяє зберігати «сесію» - адресу, порт серверу і тип підключення. Для створення сесії необхідно у головному вікні (рис. 1) ввести адресу серверу *host* в полі «*Host Name*», порт *port* в полі «*Port*» і обрати тип з'єднання *SSH*. В полі «*Saved Sessions*» необхідно ввести назву сесії (довільне ім'я) і натиснути кнопку «*Save*». Введене символьне ім'я з'явиться в списку нижче і відтепер завжди можна завантажити збережені параметри, обравши зі списку відповідне символьне ім'я і натиснувши кнопку «*Load*». Після завантаження параметрів для встановлення з'єднання з сервером необхідно натиснути кнопку «*Open*». Після цього має відкритись вікно консолі (рис. 2).

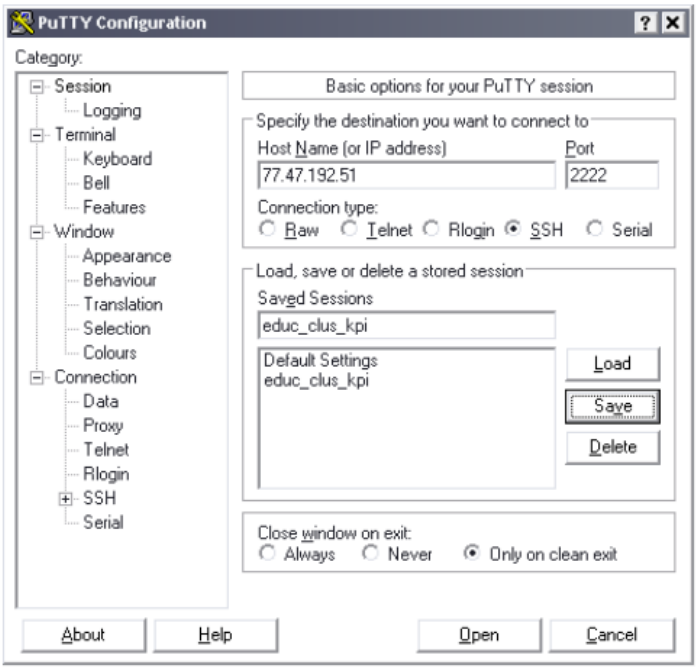


Рис. 1. Головне вікно *PuTTY*.

Рис. 2. Вікно консолі.

Якщо консоль не з’явилась, а натомість програма повідомила про помилку, то це означає, що або не працює мережа, або не працює сервер, або ви неправильно ввели надані вам дані.

В даній консолі необхідно ввести ваш *user*, а потім *pass* (введені символи паролю не відображаються на екрані з метою забезпечення безпеки). Після цього ви отримаєте доступ до командного рядка, а поточним каталогом стане домашній каталог користувача.

1.2 Консольний варіант *PuTTY*

Розглянемо запуск консольного варіанту програми *PuTTY*: вибираємо *Пуск* → *Виконати*, набираємо в полі «*cmd*» і натискаємо клавішу *Enter*. Після цього з’явиться системна консоль (рис. 3), звідки можна запустити програму.

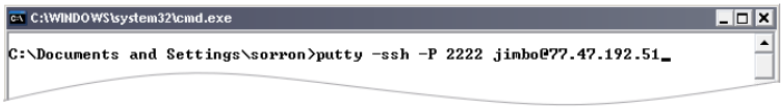


Рис. 3. Запуск *PuTTY* з командного рядка *Windows*.

Для встановлення з'єднання необхідно ввести командний рядок у наступному форматі:

```
C:\>putty -ssh -P port user@host
```

Після цього натискаємо *Enter* і з'являється вікно віддаленої консолі обчислювального вузла, де необхідно ввести пароль. Після вводу коректного паролю відбувається вхід у систему і перехід до домашньої директорії користувача:

```
login as: user
user@77.47.192.51's password:
Last login:
user@n001$ pwd
/home/user
user@n001$
```

Необхідно зауважити, що таким чином відкривається доступ до першого (головного) вузла кластеру. Для того, щоб перейти на інший вузол необхідно дати команду, в якій <*nodenumber*> – номер необхідного вузла.

```
user@n001$ ssh n<nodenumber>
```

Приклад переходу на вузол *n002*:

```
user@n001$ ssh n002
user@n002's password:
Last login:
user@n002$
```

2. Робота з файлами на кластері

Для повноцінної роботи з кластером необхідна також можливість завантажувати файли на кластер і з кластера. Для цього разом з програмою *PuTTY* встановлюється утиліта *pscp.exe* (*PuTTY secure copy client*).

Зверніть увагу! Програма *pscp.exe* працює тільки в консолі *Windows*. Необхідно виконати команду: *Пуск* → *Виконати*, набрати в полі «*cmd*» і натиснути клавішу *Enter*. З'явиться системна консоль, звідки можна запускати програму *pscp.exe*.

2.1 Перегляд списку файлів

Для виведення списку файлів в заданій директорії на обчислювальному вузлі призначена команда консолі *Windows*:

```
C:\>pscp -P 2222 -ls user@host:directory
```

Після вводу кожної команди відбувається запит паролю користувача.

Приклад виконання команди:

```
C:\>pscp -P 2222 -ls user@77.47.192.51:/home/user
user@77.47.192.51's password:
Listing directory /home/user
drwx------ 4 user user 4096 Apr 21 20:14 .
drwxr-xr-x 83 root root 4096 Apr 26 13:22 ..
-rw------- 1 user user 432 Apr 27 06:46 .bash_history
-rw-r--r-- 1 user user 33 Apr 20 11:54 .bash_logout
-rw-r--r-- 1 user user 176 Apr 20 11:54 .bash_profile
-rw-r--r-- 1 user user 124 Apr 20 11:54 .bashrc
drwx------ 2 user user 4096 Apr 21 19:18 .ssh
-rw------- 1 user user 704 Apr 21 20:14 .viminfo
drwxr-xr-x 2 user user 4096 Apr 21 20:14 gmptest
C:\>
```

2.2 Копіювання файлів на кластер

Для копіювання одного файлу на обчислювальний вузол введіть команду в консолі *Windows*:

```
C:\>pscp -P port localfile user@host:file
```

В даній команді *localfile* – ім'я файлу на локальному комп'ютері, *file* – ім'я файлу на обчислювальному вузлі. Приклад виконання команди:

```
C:\>pscp -P 2222 C:\foo.txt user@77.47.192.51:/home/user/foo.txt
user@77.47.192.51's password:
foo.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\>
```

Перевіримо наявність файлу на обчислювальному вузлі командою *ls*:

```
user@n001$ ls
foo.txt gmptest
user@n001$
```

Також на кластер можна скопіювати каталог і всі його підкаталоги. Для цього слід додати параметр *-r*:

```
C:\>pscp -P port -r localdirectory user@host:directory
```

В даній команді *localdirectory* – ім'я директорії на локальному комп'ютері, яка буде скопійована, *directory* – ім'я директорії на сервері, в яку буде скопійовано локальну директорію.

Приклад:

```
C:\dir>dir
Содержимое папки C:\dir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo1.txt
27.04.2010 07:23 <DIR> innerdir
1 файлов 46 байт
3 папок 1 773 776 896 байт свободно
C:\dir>cd innerdir
C:\dir\innerdir>dir
Содержимое папки C:\dir\innerdir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo2.txt
1 файлов 46 байт
2 папок 1 773 776 896 байт свободно
C:\dir\innerdir>pscp -P 2222 -r C:\dir user@77.47.192.51:/home/user/
user@77.47.192.51’s password:
foo1.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
foo2.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\dir\innerdir>
```

Перевіримо наявність каталогу на обчислювальному вузлі командою *ls:*

```
user@n001:~$ ls
dir foo.txt gmptest
user@n001:~/dir$ cd dir
user@n001$ ls
foo1.txt innerdir
user@n001:~/innerdir$ cd innerdir
user@n001:~/innerdir$ ls
foo2.txt
user@n001:~/innerdir$
```

2.3 Копіювання файлів з кластеру

Для копіювання файлів з обчислювального вузла на локальний комп’ютер призначена команда:

```
C:\>pscp -P port user@host:file localfile
```

Приклад роботи команди:

```
C:\dir>pscp -P 2222 user@77.47.192.51:/home/user/dir/innerdir/foo2.txt
C:\dir\foo3.txt
user@77.47.192.51’s password:
foo3.txt | 0 kB | 0.0 kB/s | ETA: 00:00:00 | 100%
C:\dir>dir
Содержимое папки C:\dir
27.04.2010 07:23 <DIR> .
27.04.2010 07:23 <DIR> ..
27.04.2010 07:23 46 foo1.txt
27.04.2010 07:51 46 foo3.txt
27.04.2010 07:23 <DIR> innerdir
2 файлов 92 байт
```

```
3 папок 1 773 776 896 байт свободно
C:\dir>
```

Література

5. Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко– К., 2011. – 154 с.
6. Жуков І.А. Паралельні та розподілені обчислення [Текст] / І.А. Жуков, О.В. Корочкін. – Київ: «Корнійчук», 2014. – 284 с.
7. Інструкція по використанню кластеру НТУУ «КПІ ім. Ігоря Сікорського» [Електронний ресурс]. – Режим доступу: http://hpcc.kpi.ua/uk/.
8. Г.И.Шпаковский, В.И.Стецюренко, А.Е. Верхотуров, Н.В. Серикова. Применение технологии MPI в Грид. – Минск, БГУ, 2008 , - 137 с.

4. 2. ***Компіляція та запуск паралельних програм на кластері***

Кластери – багатопроцесорні ЕОМ, побудовані із елементів серійного виробництва. Це відноситься не тільки до апаратури, але і до програмного забезпечення. Основною операційною системою для кластерів являється безплатна ОС Linux.

Компіляція та запуск паралельних програм на кластері

1 Система управління чергою задач

1.1. Призначення системи управління чергою задач

Під час запуску програм на своєму персональному комп’ютері Ви точно знаєте, скільки програм вже запущено та скільки вони використовують ресурсів комп’ютера. Виходячи з цієї інформації, Ви вирішуєте, коли і скільки програм запускати.

Запуск програм на кластері відрізняється від запуску програм на персональному комп’ютері: Ви не знаєте, скільки та яких ресурсів використовується в даний час на кожному з вузлів кластеру. Тому Ви не можете самостійно обрати, на яких вузлах кластеру запускати свої програми. Для вирішення цієї проблеми використовується *система управління чергою завдань*. Ідея такої системи полягає в наступному: всі користувачі кластеру повідомляють системі управління чергою, які програми вони бажають

запустити та скільки обчислювальних ресурсів цим програмам потрібно. Заявки від користувачів кластеру заносяться в чергу. Система управління чергою самостійно запускає програми з черги тоді, коли може задовольнити потреби програми в ресурсах. Таким чином, система управління чергою має повну інформацію про те, які ресурси кластеру зайняті (та якими саме програмами яких користувачів), а які ресурси кластеру вільні. Виходячи зі сказаного вище, можна зробити висновок: для продуктивної роботи всіх користувачів кластеру необхідно (та в інтересах користувачів) запускати обчислювальні програми на кластері тільки через систему управління чергою задач.

Після того, як Ви отримали доступ до командного рядка кластеру за допомогою протоколу *SSH*, Ви можете виконувати команди на вузлі *n001*. На цьому вузлі дозволяється виконувати переміщення та копіювання даних, компіляцію програм, запуск обчислювальних програм за допомогою системи управління чергою. На інших вузлах дозволяється виконувати лише інформаційні команди під час визначення причин проблем роботи програм.

На кластері Центру суперкомп'ютерних обчислень НТУУ «КПІ ім. Ігоря Сікорського» встановлена реалізація системи управління чергою задач *PBS* (англ. *Portable Batch System*) *Torque*. Нижче розглянуті основні команди для роботи з *Torque*.

1.2 Команда *qsub*

Команда *qsub* призначена для додавання програми в чергу виконання. Після успішного виконання програма видає на консоль ідентифікатор задачі в системі управління чергами (не плутати з рангом задачі в *MPI* або ідентифікатором задачі в *OpenMP*). Цей ідентифікатор знадобиться для відстежування стану виконання програми, або для видалення програми з черги.

```
qsub скрипт_старту_програми
```

скрипт_старту_програми – це ім'я файлу, в якому вказується деяка інформація для системи управління чергою задач та команди для запуску програми. Цей файл має наступний синтаксис:

```
#!/bin/bash
#PBS -S /bin/bash
```

```
#PBS -N <назва задачі>
#PBS -l <специфікація ресурсів>
#PBS -o <ім’я файлу для виводу stdout>
#PBS -e <ім’я файлу для виводу stderr>
cd <каталог з програмою>
<команда запуску програми>
```

Призначення рядків цього файлу наступне:

-S /bin/bash – параметр вказує планувальнику, що команди запуску програми слід виконувати за допомогою оболонки /bin/bash (краще не змінюйте цей параметр).

-N <назва задачі> – параметр задає назву задачі. Назва буде відображатись у списку задач (необов’язковий параметр).

-l <специфікація ресурсів> – параметр задає скільки та яких ресурсів необхідно виділити програмі. Різні види ресурсів розділяються комою. Можна задавати наступні види ресурсів:

- *walltime=HH:MM:SS* – необхідний час виконання програми (HH - години, MM - хвилини, SS - секунди). Це реальний час виконання програми, а не машинний. Якщо програма буде виконуватись довше заявленого часу, вона буде примусово завершена.
- *nodes=N:ppn=P* – необхідна кількість вузлів (*N*) та ядер на кожному вузлі (*P*). Всього програмі буде виділено $N \times P$ ядер.
- Хоча в *Torque* можна задавати ресурси оперативної пам’яті, на кластері Центру суперкомп’ютерних обчислень НТУУ «КПІ ім. Ігоря Сікорського» оперативна пам’ять розподіляється за принципом: гарантований *1 Гб* на кожне обчислювальне ядро.

-o <ім’я файлу для виводу stdout> – повний шлях до файлу, в який буде записано стандартний потік виводу програми (при звичайному запуску вручну така інформація виводиться на консоль; необов’язковий параметр).

-e <ім’я файлу для виводу stderr> – повний шлях до файлу, в який буде записано стандартний потік помилок програми (при звичайному запуску вручну ця інформація виводиться на консоль; необов’язковий параметр).

Якщо не вказати імена файлів для запису виводу програми, то ці файли створюються у поточному каталозі, а імена файлів починаються з назви задачі.

1.3 Команда *qstat*

Команда *qstat* призначена для спостереження за станом виконання задач. Дану команду можна виконувати без параметрів:

```
user@n001$ qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
1000.pbs                  example.sh       user            01:11:12 R batch
1001.pbs                  example42.sh     user            01:15:01 R batch
1002.pbs                  run.sh           user                   0 Q batch
```

Команда *qstat*.

В даному випадку видно, що користувач додав три задачі до черги. В стовпцях вказана наступна інформація:

- *Job id* – ідентифікатори задач;
- *Name* - ім’я скрипту старту програми або назва задачі (якщо задана).
- *Time Use* - кількість використаного процесорного часу.
- *S* - стан виконання задачі: «*Q*» - чекає в черзі, «*R*» - виконується.

Також команду *qstat* можна виконувати з параметром *-a* (виводить відносно більше інформації) чи з параметром *-f* (виводить дуже детальну інформацію).

1.4 Команда *qdel*

Команда *qdel* призначена для видалення задач із черги. Видаляти можна як задачу, яка очікує в черзі, так і задачу, яка вже виконується. В останньому випадку програма буде аварійно завершена.

```
user@n001$ qdel ідентифікатор_задачі
```

ідентифікатор_задачі – ідентифікатор задачі, який був виданий на консоль командою *qsub*. Якщо ви не пам’ятаєте цього ідентифікатору, дізнайтесь його командою *qstat*.

2 Компіляція та запуск програм на *MPI*

Нехай у вас є вихідні коди програми, написаної з використанням *MPI*. Для запуску цієї програми на кластері необхідно виконати наступні кроки:

1) Створіть окремий каталог для цієї програми. Наприклад, якщо ваш логін *user* та ви хочете створити каталог *example*, виконайте наступну команду:

```
user@n001$ mkdir /home/user/example
```

2) Завантажте у створений каталог вихідні коди програми.

3) Перейдіть у створений каталог командою *cd* (англ. *Change Directory*):

```
user@n001$ cd /home/user/example
```

4) Скомпілюйте програму. Якщо програма написана стороннім розробником, то разом із програмою можуть поставлятись інструкції для компіляції. Приклади програм на мові *C*, які написані з використанням *MPI*, компілюються наступним чином:

```
user@n001$ mpicc -W -Wall -O2 -std=c99 \
ім’я_файлу_1.c ім’я_файлу_2.c ... -o program
```

ім’я_файлу_1.c, ім’я_файлу_2.c – імена всіх файлів, які входять до складу програми; *program* – ім’я для виконуваного файлу програми, яке створить компілятор. Якщо програма написана на мові C++, необхідно використовувати компілятор *mpic++*.

5) Створіть в каталозі програми скрипт запуску для системи управління завданнями (чи створіть цей файл на вашому персональному комп’ютері та завантажте його на кластер у каталог програми). Нехай ім’я цього файлу буде */home/user/example/runjob.sh*. Вміст файлу може виглядати наступним чином:

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=7:ppn=2,walltime=02:00:00
#PBS -o /home/user/example/stdout.txt
#PBS -e /home/user/example/stderr.txt
cd /home/user/example
mpiexec ./program
```

Цей скрипт повідомляє *Torque* про те, що програмі необхідно виділити по 2 ядра на 7 вузлах, також програмі знадобиться якнайбільше 2 години часу для обчислень, вивід програми буде перенаправлений у файли */home/user/example/stdout.txt* та */home/user/example/stderr.txt*.

Зверніть увагу! Запускаючи *MPI* програми на кластері через систему управління чергою завдвнь, не треба передавати команді *mpiexec* додаткових параметрів *-n* чи *-np*, які задають кількість завдань. Ця інформація буде передана до *mpiexec* системою управління чергою завдань автоматично.

6) Створіть або завантажте на кластер вхідні дані для програми.

7) Додайте програму в чергу виконання командою *qsub:*

```
user@n001$ qsub /home/user/example/runjob.sh
```

8) Стан виконання завдання можна дізнатись за допомогою команди *qstat*.

3 Компіляція та запуск програм на *OpenMP*

Нехай у вас є вихідні коди програми, написаної з використанням *OpenMP*. Для запуску цієї програми на кластері необхідно виконати наступні кроки:

1) Створіть окремий каталог для цієї програми. Наприклад, якщо ваш логін *user* та ви хочете створити каталог *example*, виконайте наступну команду:

```
user@n001$ mkdir /home/user/example
```

2) Завантажте у створений каталог вихідні коди програми.

3) Перейдіть у створений каталог командою *cd* (англ. *Change Directory*):

```
user@n001$ cd /home/user/example
```

4) Скомпілюйте програму. Якщо програма написана стороннім розробником, разом з програмою можуть поставлятись інструкції для компіляції. Приклади програм на мові *C* з даного документу, що написані з використанням *OpenMP*, компілюються наступним чином:

```
user@n001$ gcc -W -Wall -O2 -std=c99 -fopenmp \
ім’я_файлу_1.c ім’я_файлу_2.c ... -o program
```

ім’я_файлу_1.c, *ім’я_файлу_2.c* – імена всіх файлів, що входять до складу програми; *program* – ім’я для виконуваного файлу програми, що створить компілятор. Якщо програма написана на мові *C++*, необхідно використовувати компілятор *g++*.

5) Створіть в каталозі програми скрипт запуску для системи управління задачами (чи створіть цей файл на вашому персональному комп’ютері та завантажте його на кластер у каталог програми). Нехай ім’я цього файлу буде */home/user/example/runjob.sh*. Вміст файлу може виглядати наступним чином:

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l nodes=1:ppn=8,walltime=02:00:00
#PBS -o /home/user/example/stdout.txt
#PBS -e /home/user/example/stderr.txt
cd /home/user/example
export OMP_NUM_THREADS=8
./program
```

Цей скрипт повідомляє *Torque* про те, що програмі необхідно виділити 8 ядер на одному вузлі, програмі також знадобиться якнайбільше 2 години часу для

обчислень, вивід програми буде перенаправлений у файли */home/user/example/stdout.txt* та */home/user/example/stderr.txt*. Значення змінної *OMP_NUM_THREADS* має бути рівне кількості ядер, вказаних в параметрі *ppn=P*.

Зверніть увагу! Запускаючи *OpenMP* програми на кластері через систему управління чергою задач, не слід виділяти програмі більше одного вузла, оскільки програми на *OpenMP* працюють тільки на системах зі спільною пам'яттю.

6) Створіть або завантажте на кластер вхідні дані для програми.
7) Додайте програму в чергу виконання командою *qsub*:

```
user@n001$ qsub /home/user/example/runjob.sh
```

8) Стан виконання завдання можна дізнатись за допомогою команди *qstat*.

4 Тестові запуски програм

Тестові (пробні для налагодження) запуски обчислювальних програм дозволяють виконувати завдання на інтерфейсному вузлі для уникнення очікування в черзі. Вимогою до тестових задач є використання не більше 4 ядер та час виконання – не більший 5 хвилин.

4.1 Тестовий запуск програм на *MPI*.

Виконайте завантаження програми та її вхідних даних на кластер, а також компіляцію програми. Після цього виконайте команду, де число 4 – це кількість завдань:

```
user@n001$ mpiexec -np 4 ./program
```

Далі розглянемо приклад розробки складної програми за допомогою MPI.

Для вирішення реальної математичної задачі на кластерній системі з використанням *MPI* і створення необхідної програми використаєм відповідний математичний апарат.

Візьмемо двовимірне диференційне рівняння Пуассона у часткових похідних виду:

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y).$$

Розв’язати його в аналітичному вигляді немає можливості. Одним із можливих методів розв’язання є застосування методу кінцевих різниць. Метод кінцевих різниць передбачає дискретизацію диференціальних рівнянь на прямокутних координатних сітках. Для двовимірних задач елементарні комірки таких сіток є прямокутниками, а для тривимірних задач комірки становлять паралелепіпеди.

Кінцево-різничні сітки. Розглянемо одновимірну область Θ, яка являє собою відрізок $[0 , s]$. Розіб’ємо цей відрізок точками $x_i = ih$, $i = 0, 1, 2, \dots , n$ на n рівних частин довжиною $h = s/n$ кожна. Множина точок $G = \{x_i = ih | i = 0, 1, 2, \dots , n\}$ називається рівномірною одномірною координатною сіткою, а число h – кроком сітки.

Відрізок $[0, s]$ можна розбити на n частин, вводячи довільні точки $0 < x_1 < x_2 < \dots < x_{i-1} < x_i < x_{i+1} < \dots < x_{n-1} < s$.

Координатна сітка $G = \{x_i | i = 0, 1, 2, \dots , n, x_0 = 0, x_n = s\}$ буде мати крок $h_i = x_i - x_{i-1}$, що залежить від номера i вузла x_i. Якщо $h_i = h_{i+1}$ хоча б для одного номера i, координатна сітка G називається нерівномірною (рис. 1).

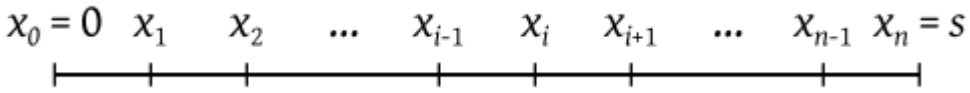


Рис. 1. Одновимірна сітка.

Аналогічно, двовимірною сіткою називають множину точок $G = \{(x_i = ih_1, y_j = jh_2) \mid i = 0, 1, 2, \dots , n; j = 0, 1, 2, \dots , m\}$. Рівномірна двовимірна сітка являє собою множину $G = \{(x_i, y_j) \mid i = 0, 1, 2, \dots , n; j = 0, 1, 2, \dots , m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$ (рис. 2).

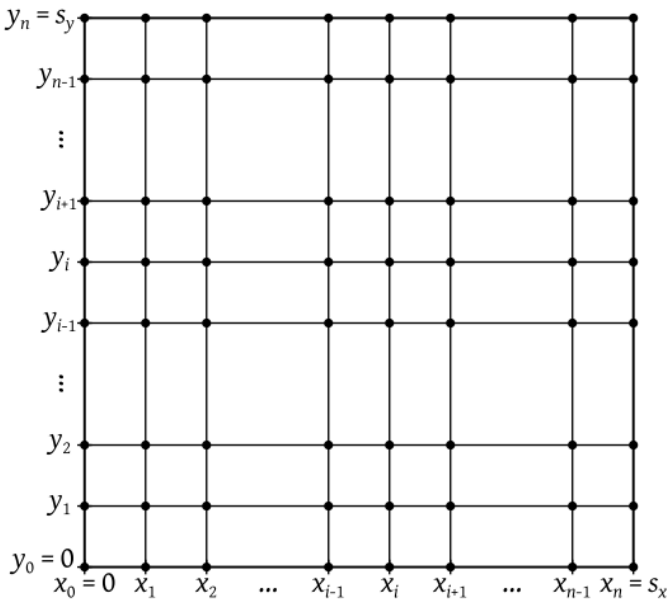


Рис. 2. Двовимірна сітка.

Сіткові функції, кінцеві різниці. Введення для області Θ координатної сітки G передбачає що значення всіх змінних та їх похідних розглядаються тільки у вузлах цієї сітки. З метою виконання цієї умови всі змінні задані сітковими функціями, а похідні будь-якого порядку - кінцевими різницями.

Нехай для деякої області Θ задана сітка $G = \{(x_i, y_j) \mid i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$. Тоді функцію $\Phi = \Phi(x_i, y_j)$, $i = 0, 1, 2, \dots, n, j = 0, 1, 2, \dots, m$ дискретного аргументу (x_i, y_j) називають сітковою функцією, визначеною на сітці G.

Будь-якій неперервній функції $f(x, y)$, заданій в області Θ, можна поставити у відповідність сіткову функцію $\Phi(x_i, y_j)$ (для зручності будемо позначати Φ_{ij}), задану на сітці $G = \{(x_i, y_j) \mid i = 0, 1, 2, \dots, n; j = 0, 1, 2, \dots, m; x_0 = 0, x_n = s_x, y_0 = 0, y_m = s_y\}$ (спроектувати функцію $f(x, y)$ на сітку G), беручи до уваги певне правило співвідношення. Наприклад:

$$\Phi_{ij} = f(x_i, y_j)$$

$$\Phi_{ij} = \frac{1}{(x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}})(y_{i+\frac{1}{2}} - y_{i-\frac{1}{2}})} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{i-\frac{1}{2}}}^{y_{i+\frac{1}{2}}} f(x, y) dx dy,$$

де $x_{i\pm\frac{1}{2}}, y_{i\pm\frac{1}{2}}$ – координати серединних точок на відповідних кроках координатної сітки, що визначаються виразами:

$$x_{i+\frac{1}{2}} = \frac{x_{i+1} + x_i}{2}$$
$$x_{i-\frac{1}{2}} = \frac{x_i + x_{i-1}}{2}$$
$$y_{i+\frac{1}{2}} = \frac{y_{i+1} + y_i}{2}$$
$$y_{i-\frac{1}{2}} = \frac{y_i + y_{i-1}}{2}$$

Слід мати на увазі, що одна й та ж сіткова функція, задана на двох різних сітках, які мають спільні вузли, не обов'язково буде мати в цих вузлах рівні значення. За визначенням похідна функції неперервного аргументу x у точці x_0 є ліміт відношення приросту функції до приросту аргументу, коли приріст аргументу наближається до нуля:

$$\frac{\partial f(x)}{\partial x} = \lim_{(x-x_0)\to 0} \frac{f(x) - f(x_0)}{x - x_0}$$

Знехтувавши границею, похідну функції неперервного аргументу можна приблизно замінити (апроксимувати) різницевим виразом, заданим на відповідній сітковій функції $\Phi(x_i,y_j)$. Дана апроксимація може бути виконана декількома способами:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{\Phi_{i+1,j} - \Phi_{i,j}}{\Delta x_i},$$
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{\Phi_{i,j} - \Phi_{i-1,j}}{\Delta x_i},$$
$$\frac{\partial f(x,y)}{\partial y} \approx \frac{\Phi_{i,j+1} - \Phi_{i,j}}{\Delta y_j},$$
$$\frac{\partial f(x,y)}{\partial y} \approx \frac{\Phi_{i,j} - \Phi_{i,j-1}}{\Delta y_j},$$

де $\Delta x_i, \Delta y_j$ – кінцеві різниці координат, що визначаються виразами:

$$\Delta x_i = x_{i+1} - x_i,$$
$$\Delta y_i = y_{j+1} - y_j.$$

Для кожного з перетворень характерна така похибка апроксимації, яка прямує до нуля коли крок сітки прямує до нуля.

Похідні другого порядку апроксимуються наступним чином:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{\frac{\Phi_{i+1,j}-\Phi_{i,j}}{\Delta x_i} - \frac{\Phi_{i,j}-\Phi_{i-1,j}}{\Delta x_{i-1}}}{\frac{\Delta x_i + \Delta x_{i-1}}{2}}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx \frac{\frac{\Phi_{i,j+1}-\Phi_{i,j}}{\Delta y_i} - \frac{\Phi_{i,j}-\Phi_{i,j-1}}{\Delta y_{j-1}}}{\frac{\Delta y_i + \Delta y_{i-1}}{2}}$$

У випадку, коли сітка рівномірна, вирази спрощуються:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \approx \frac{\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j}}{\Delta x^2}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \approx \frac{\Phi_{i,j+1} - 2\Phi_{i,j} + \Phi_{i,j-1}}{\Delta y^2}.$$

Аналіз задачі

Застосувавши вище приведений метод до нашої задачі, ми можемо визначити структуру обчислювальної моделі та формули, за якими будемо вести розрахунок. Структура обчислювальної моделі зображена на рис. 3. Сітка дискретизації квадратна та рівномірна. У кожному обчислювальному вузлі буде послідовно розраховуватися матриця точок сітки дискретизації. Реальні вузли будуть працювати паралельно між собою та обмінюватися даними.

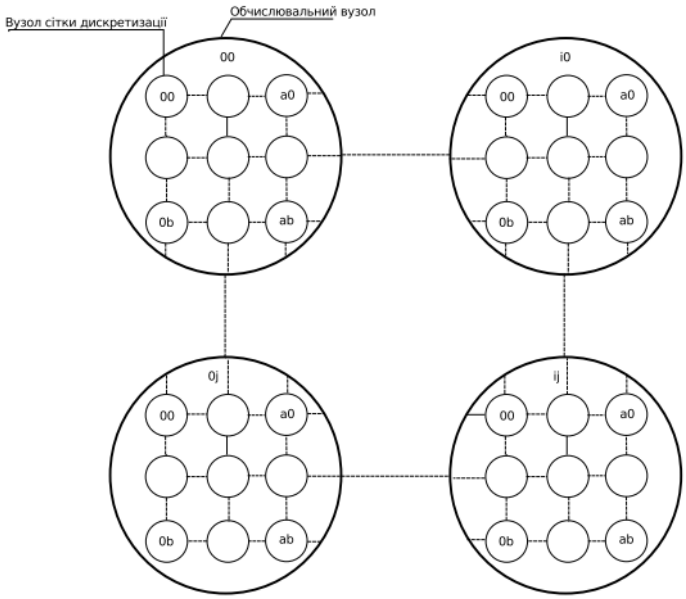


Рис. 3. Структура обчислювальної моделі.

Нехай початкові умови функції всередині області $U_0 = 0$, а на границі області $U_0 = (1 + 3\pi) \sin(3\pi(x + y))$.

$$U_{nm}(z) = U_{nm}(z-1) - \omega\frac{h^2}{4}(LU_{nm} - f_{nm}),$$

де:

- n, m – координати вузла сітки дискретизації;
- z – номер ітерації;
- h – крок у просторі, що визначається за формулою $h = 1/N$;
- N – кількість вузлів сітки дискретизації вздовж однієї з координат;
- f_{nm} – значення правої частини диференційного рівняння у вузлі сітки дискретизації;
- LU_{nm} – різницевий оператор, що визначається за формулою:

$$LU_{nm} = U_{nm} - \frac{U_{n+1,m} - 2U_{n,m} + U_{n-1,m}}{h^2} - \frac{U_{n,m+1} - 2U_{n,m} + U_{n,m-1}}{h^2}.$$

Локальним критерієм завершення алгоритму слугує досягнення заданої точності:

$$\frac{U_{nm}(z) - U_{nm}(z-1)}{U_{nm}(z)} \leq \varepsilon,$$

де ε задається до початку розрахунку.

Виконання

Спрощений алгоритм основного циклу програми виглядає наступним чином:

1. Зробити розрахунки для тих вузлів сітки дискретизації, які не потребують даних від сусідніх процесів.
2. Отримати колонку сітки дискретизації від лівого сусіднього процесу, якщо він є.
3. Передати крайню праву колонку сітки дискретизації до правого сусіднього процесу, якщо він є.
4. Отримати колонку сітки дискретизації від правого сусіднього процесу, якщо він є.

5. Передати крайню ліву колонку сітки дискретизації до лівого сусіднього процесу, якщо він є.
6. Отримати рядок сітки дискретизації від верхнього сусіднього процесу, якщо він є.
7. Передати нижній рядок сітки дискретизації до нижнього сусіднього процесу, якщо він є.
8. Отримати рядок сітки дискретизації від нижнього сусіднього процесу, якщо він є.
9. Передати верхній рядок сітки дискретизації до верхнього сусіднього процесу, якщо він є.
10. Завершити розрахунки для інших вузлів, які не були розраховані у пункті 1.
11. Перевірити локальні критерії завершення алгоритмів. Якщо всі вузли сітки у процесі завершили свою роботу- завершити розрахунки у процесі.
12. Якщо всі процеси завершили розрахунки - завершити алгоритм.

Розглянемо деякі проблеми, зв'язані з реалізацією цього алгоритму:

1) В мові *C* матриці зберігаються по рядках (елементи одного рядку розташовані послідовно в пам'яті). Алгоритм розв'язання задачі передбачає передачу рядків та стовпців матриці. Елементи одного рядку можна передати за допомогою звичайного виклику *MPI_Send*, оскільки вони розташовані послідовно. Елементи одного стовпця таким чином неможливо передати, тому що вони розташовані в пам'яті з кроком, рівним ширині матриці. Для передачі стовпців матриці створимо тип «стовпець матриці» за допомогою функції конструктора типу *MPI_Type_vector* наступним чином:

```
MPI_Type_vector( <висота матриці>,
1, /* довжина одного блоку */
<ширина матриці>,
MPI_DOUBLE,
&column_type);
```

Для передачі стовпця матриці цей тип можна застосувати наступним чином:

```
MPI_Send( <Покажчик на перший елемент стовпця>,
1, /* кількість стовпців */
column_type,
<ранг задачі- отримувача>,
<тег>,
MPI_COMM_WORLD);
```

2) Аналогічна проблема виникає при початковому розподілі координат вузлів сітки дискретизації по процесах. Фактично нам потрібно відправити квадратну під-матрицю. Реалізація змінюється тільки у ширині блоку і їх кількості при конструюванні типу.

```
MPI_Type_vector( <висота блоку, що відправляємо>,
<ширина блоку, що відправляємо>,
<ширина матриці>,
MPI_DOUBLE,
&new_type);
```

3) Попри те, що процес закінчив розрахунок, він усе одно повинен взаємодіяти з сусідніми процесами, щоб ті теж могли завершити розрахунки. Для цього треба реалізувати подвійну перевірку – умови зупинки розрахунку процесу та умови глобального кінця розрахунку. У програмі спочатку виконується перевірка усіх локальних умов закінчення алгоритму. Якщо усі вузли сітки дискретизації, що належать до процесу, закінчили роботу, то встановлюється ознака закінчення обчислень у процесі. Після цього усі ознаки закінчення обчислень у процесах збираються нульовим процесом і над ними виконується операція логічного «*І*» за допомогою *MPI_Reduce*.

```
MPI_Reduce( &node_stop,
&global_stop,
1,
MPI_SHORT,
MPI_LAND,
0,
MPI_COMM_WORLD);
```

Результат цієї операції є глобальною ознакою закінчення алгоритму, оскільки він матиме значення *true* тільки коли усі процеси закінчили обчислення. Після цього глобальна ознака закінчення обчислення розповсюджується між усіма процесами за допомогою *MPI_Bcast*.

Код програми

Файл *code/lab_diffeq/main.c*

```
#define _GNU_SOURCE
#include <mpi.h>
#include <math.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
```

```
#include <stdbool.h>
/* Опис типів, структур та функцій */
struct my_matrix /* Структура, що описує матриці */
{
int rows; /* Кількість строк у матриці */
int cols; /* Кількість стовпців у матриці */
double *data; /* Покажчик на матрицю, розвернуту у одномірний масив*/
};
MPI_Datatype node_mat_t; /* Тип для розповсюдження частин сітки
дискретизації по матриці процесів*/
MPI_Datatype mat_col; /* Тип для передачі стовпця матриці */
void inline evaluate( int); /*Функція, що містить ітераційні формули */
double func( double X, double Y); /* Повертає значення правої частини
диференційного рівняння у точці X,Y*/
void init_grid(); /* Ініціалізує значення сітки дискретизації у процесі */
void fatal_error( const char *message, int errorcode); /* Функція виведення
помилок вводу-виводу */
struct my_matrix *matrix_alloc( int rows, int cols, double initial); /* Функція
для виділення пам'яті та ініціалізацію матриці*/
void matrix_print( const char *filename, struct my_matrix *mat); /* Функція
для виведення матриці у файл*/
struct my_matrix *read_matrix( const char *filename); /* Функція
для зчитування матриці із файлу*/
/* ********************************************************************** */
/* Опис змінних */
int np; /* Кількість процесів */
int rank; /* Ранг процесу у MPI_COMM_WORLD */
int nodes_width; /* Ширина матриці процесів */
int node_X; /* Координата X у матриці процесів */
int node_Y; /* Координата Y у матриці процесів */
int total_width; /* Ширина матриці вузлів сітки дискретизації*/
int points_per_node; /* Ширина частини матриці вузлів сітки дискретизації,
що припадає на кожен процес*/
const char* inpfileX="inpX.csv"; /* Ім'я файлу з якого вводяться горизонтальні
координати сітки дискретизації */
const char* inpfileY="inpY.csv"; /* Ім'я файлу з якого вводяться вертикальні
координати сітки дискретизації */
const char* inpfileInit="inpInit.csv"; /*Ім'я файлу з якого вводяться початкові та
граничні умови для сітки дискретизації*/
const char* outpfile="outp.csv"; /* Ім'я файлу куди виводиться результат */
my_matrix* allcoords_X; /* Матриця реальних горизонтальних координат вузлів
сітки дискретизації */
my_matrix* allcoords_Y; /* Матриця реальних вертикальних координат вузлів сітки
дискретизації */
my_matrix* all_grid_init; /*Всі початкові та граничні умови функції*/
my_matrix* total_mat; /* Повна сітка дискретизації, що виводиться у процесі 0*/
my_matrix* grid_init; /*Частина початкових умов для процесу*/
my_matrix* coords_X; /* Частина матриці реальних координат сітки дискретизації,
яка зберігається у кожному з процесів*/
my_matrix* coords_Y; /* Частина матриці реальних координат сітки дискретизації,
яка зберігається у кожному з процесів*/
my_matrix* node_mat; /* Частина сітки дискретизації, яка зберігається у кожному з
процесів*/
my_matrix* f_xy; /* Масив значень функції в вузлах сітки дискретизації*/:
MPI_Status stat; /* Змінна, у яку повертається статус прийому повідомлень */
double epsilon=0.01; /* Точність рішення*/
double omega=0.4; /* Коефіцієнт релаксації*/
int max_iter=1000000; /*Максимальна кількість ітерацій у випадку нев'язки функції
*/
double h; /* Крок сітки дискретизації */
bool* local_stop; /* Масив для запам'ятовування локальних умов зупинки */
int* node_iter;
short node_stop; /* Ознака завершення обчислень у процесі*/
```

```
short global_stop; /* Глобальна ознака завершення обчислень*/
double* left_col; /* Стовпці, що процес буде приймати з лівого та правого
процесів під час ітерацій */
double* right_col;
double* top_row; /* Рядки, що процес буде приймати з верхнього та нижнього
процесів під час ітерацій */
double* bot_row;
/* ********************************************************************** */
 /* Головна програма */
int main( int argc, char *argv[])
{
MPI_Init( &argc, &argv); /* Ініціалізуємо середовище */
MPI_Comm_size( MPI_COMM_WORLD, &np); /* Отримуємо розмір MPI_COMM_WORLD */
MPI_Comm_rank( MPI_COMM_WORLD, &rank); /* Отримуємо ранг процесу у
MPI_COMM_WORLD*/
nodes_width=sqrt(np); /* Отримуємо ширину матриці процесів*/
node_X=rank%nodes_width; /* Отримуємо горизонтальну координату процесу у матриці
*/
node_Y=rank/nodes_width; /* Отримуємо вертикальну координату процесу у матриці */
if (rank==0)
{
allcoords_X=read_matrix(inpfileX); /* Вводимо координати */
allcoords_Y=read_matrix(inpfileY); /* Вводимо координати */
all_grid_init=read_matrix(inpfileInit); /* Вводимо координати */
total_width=allcoords_X- >rows; /* Отримуємо ширину сітки дискретизації */
}
MPI_Bcast( &total_width, 1, MPI_INT, 0, MPI_COMM_WORLD); /* Розповсюджуємо ширину
сітки дискретизації */
h=1.0/total_width; /* Визначаємо крок сітки дискретизації */
points_per_node=total_width/nodes_width;
/* Створюємо тип, що відповідає частині матриці, яка буде передаватися */
MPI_Type_vector(points_per_node,points_per_node,total_width, MPI_DOUBLE,
&node_mat_t);
MPI_Type_commit( &node_mat_t); /* Реєструємо тип */
coords_X=matrix_alloc(points_per_node,points_per_node, 0.0);
coords_Y=matrix_alloc(points_per_node,points_per_node, 0.0);
grid_init=matrix_alloc(points_per_node,points_per_node, 0.0);
if(rank==0)
{/* Процес 0 розсилає реальні координати сітки дискретизації*/
for( int i=0;i<nodes_width;i++)
{
for( int j =0;j <nodes_width;j ++)
{
if( ! (i==0&&j ==0))
{
MPI_Send(allcoords_X- >data+i*total_width*points_per_node+j *points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
MPI_Send(allcoords_Y- >data+i*total_width*points_per_node+j *points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
MPI_Send(all_grid_init- >data+i*total_width*points_per_node+j *points_per_node
, 1,node_mat_t,i*nodes_width+j, 0, MPI_COMM_WORLD);
}
}
}
for( int i=0;i<points_per_node;i++)
{
for( int j =0;j <points_per_node;j ++)
{
coords_X- >data[i*points_per_node+j] =allcoords_X- >data[i*total_width+j];
 coords_Y- >data[i*points_per_node+j] =allcoords_Y- >data[i*total_width+j];
grid_init- >data[i*points_per_node+j] =all_grid_init- >data[i*total_width+j];
}
}
```

```
}
else
{/* Інші процеси їх приймають */
my_matrix* temp_X=matrix_alloc(total_width,points_per_node, 0);
my_matrix* temp_Y=matrix_alloc(total_width,points_per_node, 0);
my_matrix* temp_init=matrix_alloc(total_width,points_per_node, 0);
MPI_Recv(temp_X- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
MPI_Recv(temp_Y- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
MPI_Recv(temp_init- >data, 1,node_mat_t, 0, 0, MPI_COMM_WORLD, &stat);
for( int i=0;i<points_per_node;i++)
{
for( int j =0;j <points_per_node;j ++)
{
coords_X- >data[i*points_per_node+j] =temp_X- >data[i*total_width+j];
coords_Y- >data[i*points_per_node+j] =temp_Y- >data[i*total_width+j];
grid_init- >data[i*points_per_node+j] =temp_init- >data[i*total_width+j];
}
}
}
/* Ініціалізуємо умови локальної зупинки алгоритму*/
local_stop=(bool*)malloc(points_per_node*points_per_node*sizeof(bool));
for( int i=0;i<points_per_node*points_per_node;i++)
{
local_stop[i] =false;
}
for( int i=0;i<points_per_node;i++) /* Граничні точки залишаються постійними*/
{
if(node_Y==0)
{
local_stop[i] =true;
}
if(node_X==0)
{
local_stop[i*points_per_node] =true;
}
if(node_X==nodes_width- 1)
{
local_stop[(i+1) *points_per_node- 1] =true;
}
if(node_Y==nodes_width- 1)
{
local_stop[points_per_node*(points_per_node- 1) +i] =true;
}
}
/* Ініціалізуємо і встановлюємо початкові значення внутрішнього діапазону */
node_mat=matrix_alloc(points_per_node,points_per_node, 0);
/* Встановлюємо початкові значення граничних вузлів сітки дискретизації */
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_mat- >data[i] =grid_init- >data[i];
}
/* Створимо масив значень функції в локальних вузлах сітки дискретизації*/
f_xy=matrix_alloc(points_per_node,points_per_node, 0.0);
for( int i=0;i<points_per_node*points_per_node;i++)
{
f_xy- >data[i] =func(coords_X- >data[i],coords_Y- >data[i]);
}
/*Визначимо типи для передач під час ітерацій*/
/* Тип, що визначає стовпець матриці*/
MPI_Type_vector(points_per_node, 1,points_per_node, MPI_DOUBLE, &mat_col);
MPI_Type_commit( &mat_col); /* Реєструємо тип */
left_col=( double*)malloc(points_per_node*points_per_node*sizeof( double));
right_col=( double*)malloc(points_per_node*points_per_node*sizeof( double));
```

```
top_row=( double*)malloc(points_per_node*points_per_node*sizeof( double));
bot_row=( double*)malloc(points_per_node*points_per_node*sizeof( double));
node_iter=( int*)malloc(points_per_node*points_per_node*sizeof( int));
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_iter[i] =0;
}
/* Початок ітерацій */
do
{
/* Обмін між процесами у сітці*/
if(node_X! =0)
{
/* Приймемо стовпець з лівого процесу */
MPI_Recv(left_col, 1,mat_col,rank- 1,rank- 1, MPI_COMM_WORLD, &stat);
}
if(node_X! =nodes_width- 1)
{
/* Відішлемо стовпець до правого процесу */
MPI_Send(node_mat- >data+points_per_node- 1, 1,mat_col
,rank+1,rank, MPI_COMM_WORLD);
/* Приймемо стовпець з правого процесу */
MPI_Recv(right_col, 1,mat_col,rank+1,rank+1, MPI_COMM_WORLD, &stat);
}
if(node_X! =0)
{
 /* Відішлемо стовпець до лівого процесу */
MPI_Send(node_mat- >data, 1,mat_col,rank- 1,rank, MPI_COMM_WORLD);
}
if(node_Y! =0)
{
/* Приймемо строку з верхнього процесу */
MPI_Recv(top_row,points_per_node, MPI_DOUBLE,rank- nodes_width
,rank- nodes_width, MPI_COMM_WORLD, &stat);
}
if(node_Y! =nodes_width- 1)
{
/* Відішлемо строку до нижнього процесу */
MPI_Send(node_mat- >data+(points_per_node- 1) *points_per_node,points_per_node
, MPI_DOUBLE,rank+nodes_width,rank, MPI_COMM_WORLD);
/* Приймемо строку з нижнього процесу */
MPI_Recv(bot_row,points_per_node, MPI_DOUBLE,rank+nodes_width,
rank+nodes_width, MPI_COMM_WORLD, &stat);
}
if(node_Y! =0)
{
/* Відішлемо строку до верхнього процесу */
MPI_Send(node_mat- >data,points_per_node, MPI_DOUBLE,
rank- nodes_width,rank, MPI_COMM_WORLD);
}
/* Виклик функції, що проводить ітераційну обробку*/
for( int i=0;i<points_per_node;i++)
{
for( int j =0;j <points_per_node;j ++)
{
evaluate(i*points_per_node+j);
}
}
/*Якщо усі вузли сітки дискретизації у процесі завершили
обчислення - встановлюємо ознаку завершення обчислень у процесі*/
node_stop=1;
for( int i=0;i<points_per_node*points_per_node;i++){
if ( ! local_stop[i]){
```

```
node_stop=0;
break;
}
}
global_stop=1;
/*Перевіряємо чи всі процеси закінчили обчислення*/
MPI_Reduce( &node_stop, &global_stop, 1, MPI_SHORT, MPI_BAND, 0, MPI_COMM_WORLD);
 /*Якщо всі - то посилаємо сигнал про завершення*/
MPI_Bcast( &global_stop, 1, MPI_SHORT, 0, MPI_COMM_WORLD);
} while( ! global_stop);
/* Збираємо результат */
total_mat=matrix_alloc(total_width,total_width, 0.0);
if(rank==0){
for( int i=0;i<nodes_width;i++){
for( int j =0;j <nodes_width;j ++){
if( ! (i==0&&j ==0)){
my_matrix* temp=matrix_alloc(points_per_node,points_per_node, 0.0);
/* Приймаємо частини матриці сітки дискретизації з кожного процесу */
MPI_Recv(temp- >data,points_per_node*points_per_node,
MPI_DOUBLE,i*nodes_width+j,i*nodes_width+j, MPI_COMM_WORLD, &stat);
/* Та розташовуємо їх у повній матриці */
for( int k=0;k<points_per_node;k++){
for( int l=0;l<points_per_node;l++){
total_mat- >data[i*total_width*points_per_node+
j *points_per_node+k*total_width+l]
=temp- >data[k*points_per_node+l];
}
}
}
}
}
for( int i=0;i<points_per_node;i++){
for( int j =0;j <points_per_node;j ++){
total_mat- >data[i*total_width+j] =node_mat- >data[i*points_per_node+j];
}
}
}
else{
MPI_Send(node_mat- >data,points_per_node*points_per_node
, MPI_DOUBLE, 0,rank, MPI_COMM_WORLD);
}
if (rank==0){
matrix_print(outpfile,total_mat);;
}
MPI_Finalize();
}
/* ********************************************************************** */
/* Допоміжні функції */
double func( double X, double Y)
{
return X*Y;
}
/*Функція, що містить ітераційні формули */
 void inline evaluate( int index)
{
/*Якщо досягнута потрібна точність,
то не потрібно проводити обчислення */
if (local_stop[index])
{
return;
}
/*Якщо вузол сітки знаходиться на границі між обчислювальними
вузлами, то потрібно взяти значення з буфера передачі*/
```

```
double left=index%points_per_node==0
?left_col[index] : node_mat- >data[index - 1];
double right=(index + 1) %points_per_node==0
?right_col[index+1- points_per_node] : node_mat- >data[index + 1];
double top=index/points_per_node == 0 ? top_row[index % points_per_node]
: node_mat- >data[index- points_per_node];
double bottom=index/points_per_node==points_per_node- 1
?bot_row[index%points_per_node]
: node_mat- >data[index+points_per_node];
/*Власне обчислення*/
double old_node=node_mat- >data[index];
double LU=old_node- (left+right+top+bottom- 4*old_node) /(h*h);
node_mat- >data[index] =old_node- omega*h*h/4*(LU- f_xy- >data[index]);
/*Якщо досягнута точність або перевищена максимальна кількість ітерацій,
то завершуємо обчислення у цьому вузлі сітки*/
if (fabs(node_mat- >data[index] - old_node) <=
fabs(node_mat- >data[index] *epsilon) | | ++node_iter[index] >max_iter)
{
local_stop[index] =true;
}
}
/* Ініціалізує значення сітки дискретизації у процесі */
void init_grid()
{
for( int i=0;i<points_per_node*points_per_node;i++)
{
node_mat- >data[i] =grid_init- >data[i];
}
}
void fatal_error( const char *message, int errorcode)
{
printf( "fatal error: code %d, %s\n", errorcode, message);
fflush(stdout);
MPI_Abort( MPI_COMM_WORLD, errorcode);
}
struct my_matrix *matrix_alloc( int rows, int cols, double initial)
{
struct my_matrix *result = (my_matrix*)malloc( sizeof( struct my_matrix));
result- >rows = rows;
 result- >cols = cols;
result- >data = ( double*)malloc( sizeof( double) * rows * cols);
for( int i = 0; i < rows; i++)
{
for( int j = 0; j < cols; j ++)
{
result- >data[i * cols + j] = initial;
}
}
return result;
}
void matrix_print( const char *filename, struct my_matrix *mat)
{
FILE *f = fopen(filename, "w");
if(f == NULL)
{
fatal_error( "cant write to file", 2);
}
for( int i = 0; i < mat- >rows; i++)
{
for( int j = 0; j < mat- >cols; j ++)
{
fprintf(f, "%lf ", mat- >data[i * mat- >cols + j]);
}
```

```
fprintf(f, ”\n”);
}
fclose(f);
}
struct my_matrix *read_matrix( const char *filename)
{
FILE *mat_file = fopen(filename, ”r”);
if(mat_file == NULL)
{
fatal_error( ”can’t open matrix file”, 1);
}
int rows;
int cols;
fscanf(mat_file, ”%d %d”, &rows, &cols);
struct my_matrix *result = matrix_alloc(rows, cols, 0.0);
for( int i = 0; i < rows; i++)
{
for( int j = 0; j < cols; j ++)
{
fscanf(mat_file, ”%lf”, &result- >data[i * cols + j]);
}
}
fclose(mat_file);
return result;
}
```

Література

1. Засоби паралельного програмування [Текст] / С.Г. Стіренко, Д.В. Грибенко, О.І. Зіненко, А.В. Михайленко– К., 2011. – 154 с.
2. Жуков І.А. Паралельні та розподілені обчислення [Текст] / І.А. Жуков, О.В. Корочкін. – Київ: «Корнійчук», 2014. – 284 с.
3. Інструкція по використанню кластеру НТУУ «КПІ ім. Ігоря Сікорського» [Електронний ресурс]. – Режим доступу: http://hpcc.kpi.ua/uk/.

5 Розвиток суперкомп’ютерних технологій

5.1. *Криза фоннеймановської моделі обчислень і ознаки третьої суперкомп’ютерної революції.*

Час летить стрімко. Я — із тих, хто ще пам’ятає, що таке «пульт центрального процесора» і як він виглядає, але зараз в офіси комп’ютерних фірм прийшло зовсім нове покоління молодих професіоналів. Останнє слово сказано без всякої іронії. І це стосується Вас в першу чергу, бо якщо в дипломах ваших попередників років двадцять назад значилось, що вони є

інженери-електрики, то в Ваших записується, що Ви є професіоналами з програмного забезпечення. А професіонал є професіонал. Диплом кандидата наук наук не є підтвердженням його професіоналізму. Це людина, яка перш за все задовольняла «праздное любопытство за казенный сячет». Можливо зараз придумали щось інше, але пару років назад було саме так. Але запитайте у сучасних випускників щось з основ обчислювальної техніки і ви отримаєте самі різні непереконливі відповіді. Подобається це мені чи ні, але це нормально. Наш світ зовсім став іншим. Люди знають і вміють те, що їм потрібно знати і вміти. І це прекрасно, що в сучасному комп'ютері виснажливі подробиці внутрішньої будови, якими нас мучили на моладших курсах, надійно сховані за кам'яною стіною ОС і системи програмування, так що розробник може зосередитися на вирішенні виключно творчих задач. Але це добре лише там і тоді, коли ці надійно заховані технічні подробиці залишаються незмінними. Коли ж усталений, вічний і непорушний фундамент традиційних технологій починає змінюватися, тоді професіонал опиняется в ролі сільського жителя, який вміє змінити провайдера стільникового зв'язку, але не вміє ремонтувати власний трактор.

Ніщо в нашому світі не змінюється так швидко, як комп'ютерні технології. Але парадокс у тому, що їх фундамент — основи архітектури процесорів і операційних систем — жахливо консервативні. Здебільшого вони не змінюються вже більше 50 років. Саме тому їх і вдалося, зрештою, так надійно сховати. Є, однак, одна галузь комп'ютерного світу, яка саме зараз, наприкінці другого десятиліття XXI століття, стоїть перед реальною перспективою не просто швидких і значних, але глибинних змін у буквальному значенні цього слова — струсу основ. І галузь ця — високопродуктивні обчислення, все те, що пов'язане із суперкомп'ютерами. Так, масовою цю галузь, за всіма бажанням, не назвеш. Але вона існує і потребує спеціалістів, здатних відповісти на виклик часу.

Коли на зміну процесору Пепнтіум-3 приходить процесор Пепнтіум -4, особливості відмінностей між ними можна сміливо залишити спеціалістам

відповідного відділу фірми Інтел. А коли на зміну процесору приходить реконфигуруєма логічна матриця, розуміння того, що таке процесор, як і взагалі він буває влаштований, стає питанням життя чи згасання суперкомп'ютерної галузі. В таких випадках виникає потреба відшукати тих небагатьох, хто все ще пам'ятає, що там у цього процесора всередині.

І щоразу таких фахівців знайти все важче і важче. А то й не знайти зовсім, бо їх уже нема. Коли Вам доведеться почати роботу над принципово новим суперкомп'ютером, зовсім не схожим на машини зі списку Top-500, Вам не раз і не два захочеться порадитися з такими фахівцями. З тими, хто на зорі вітчизняної інформатики ще не ділив роботу комп'ютерного інженера на «математику», «системне програмування» та «електроніку», а умів бачити завдання в цілому, у всій складності взаємозв'язків, не завжди очевидних вузькому фахівцю.

Для початку спробуємо зв'язати в логічний ланцюжок такі часто вживані сьогодні поняття, як паралельна обробка даних, суперкомп'ютер, високопродуктивні обчислення, паралельна обчислювальна система. Для цього нам знадобиться відповісти на зовсім вже, здавалося б, просте питання — а що таке «звичайний комп'ютер»? Близько 70 років тому, на зорі сучасної інформатики, був придуманий певний спосіб об'єднання логічних елементів у обчислювальну систему, якою можна було користуватися, не замислюючись про фізику її функціонування.

Спосіб цей отримав назву фоннейманівської машини і виявився настільки вдалим, що практично без змін дожив до сьогодні. Ми звикли до нього настільки, що називаємо побудовані цим способом машини просто комп'ютер, звичайний комп'ютер. Фоннейманівський комп'ютер, власне кажучи — паралельна обчислювальна система. У розгорнутому формулюванні своїх знаменитих принципів [1] фон Нейман наполягав на тому, що операції над числами у його машині повинні виконуватися паралельно лише на рівні двійкових розрядів, тобто. не розряд за розрядом, а з усіма розрядами відразу. На той час це був важкодосяжний ідеал. Необхідну для такої організації

арифметичного пристрою кількість ламп і діодів важко було зібрати в одну схему і змусити працювати без помилок. Так чи інакше, порозрядний паралелізм було закладено в архітектуру вже тим фактом, що операції над числами задавалися як єдині, неподільні машинні команди. Минали роки, і всі настільки звикли до фоннеймановського комп'ютера, що перестали вважати його паралельним — закладений у нього рівень паралелізму став розумітися як щось, само собою зрозуміле. Кількість логічних елементів, технологічно доступних для об'єднання в єдину обчислювальну систему, з часів фон Неймана донині зросла в мільярди разів, і це зростання продовжується. Приблизно наприкінці 80-х років XX ст. фоннеймановські принципи, залишаючись фундаментальними, стали недостатніми. Математики ж вимагали від розробників обчислювальних систем дедалі більшої та більшої швидкодії. Виникла нагальна потреба доповнити, «надбудувати» фоннеймановські принципи, тобто навчитися будувати машини набагато «паралельніші», ніж фоннейманівський комп'ютер. Саме такі системи отримали назву паралельних обчислювальних систем, а способи їх побудови та використання стали називатися технологіями паралельної обробки даних. Щоб підкреслити відмінність новостворених комп'ютерів від традиційних машин, останні стали називати послідовними комп'ютерами, що, строго кажучи, не зовсім точно, але відображає факт послідовного виконання окремих команд, із яких складається програма.

Тут треба зазначити, що ще задовго до того, як виникла нагальна необхідність доповнення фоннеймановських принципів, комп'ютери розділилися на машини масового (за поняттями, звичайно) виробництва і машини рекордної продуктивності, які випускаються окремими екземплярами або малими серіями. Останні стали називати суперкомп'ютерами, а способи їх побудови та використання — технологіями високопродуктивних обчислень. Спочатку ці технології розвивалися в рамках фоннейманівських принципів, але в кінці 80-х, коли це перестало бути можливим, суперкомп'ютери стали паралельними обчислювальними системами, а технології високопродуктивних обчислень стали невід'ємними від технологій паралельної обробки даних.

Важливо відзначити, що «мотором» розвитку суперкомп'ютерних (і взагалі комп'ютерних) архітектур є саме зростання кількості доступних для об'єднання в систему логічних елементів (часто говорять про кількість транзисторів, маючи на увазі, що кожен логічний елемент будується з певної і не надто великої кількості транзисторів). Радикально нова архітектура може бути придумана і може навіть пройти стадію дослідних зразків, але фактом комп'ютерної промисловості вона стає тоді, коли зростання кількості доступних елементів перестає «вміщатися» в стару архітектуру. Кожен такий технічний переворот просто болісний для користувачів, тому йдуть на нього лише тоді, коли всі засоби роботи в старих рамках дійсно вичерпані.

Останні роки технології паралельної обробки даних почали застосовуватися у класі комп'ютерів масового випуску. Про це ми поговоримо пізніше. Поки ж вважатимемо, що комп'ютер масового випуску — це послідовна (фоннеймановська) машина, а суперкомп'ютер — це паралельна обчислювальна система, і зосередимося на суперкомп'ютерах.

У своєму розвитку паралельні суперкомп'ютери пройшли два чітко помітні етапи. На першому етапі вони будувалися, в основному, як системи з великої кількості інтегрованих в єдину систему послідовних машин, або послідовних машин з невеликими, чітко окресленими паралельними розширеннями, такими, як команди векторної обробки даних. На цьому етапі існує безліч різних способів побудови системи із складових її вузлів, але вузлом у будь-якому випадку є «звичайний» або «майже звичайний» комп'ютер. Даний етап був тривалим і в даний час завершується. Завершення першого та перехід до другого етапу обумовлені тими самими причинами, що й перехід до технологій паралельної обробки даних у класі комп'ютерів масового випуску. Межа кількості елементів, придатних для «упаковки» у фоннеймановську архітектуру, нині вже перекрито не тільки у межах обчислювальної системи загалом, а й у рамках окремого мікропроцесора. Для перетворення цієї неймовірної кількості елементів на реально досяжну обчислювальну швидкодію вже мало архітектури з так чи інакше взаємодіючих фоннеймановських вузлів. Потрібно змінювати «конструкцію» самого вузла. Процес появи відповідних технологій

вже розпочався. Майже напевно його результати за своїми революційністю та несподіванками перевершать усі мислимі та немислимі очікування та прогнози, причому найближчим часом.

Освоєні суперкомп’ютерні технологии ми коротко розглянули в попередніх лекціях.

Що стосується перспективи, то раптові радикальні зміни не за горами. У чому ж вони полягатимуть? Згадаймо, що нам відомо про закономірності зміни вигляду суперкомп'ютерів:

1) вигляд суперкомп'ютера на кожному етапі технологічного розвитку визначається Правилом економії сил розробника. Суперкомп'ютери просто змушені бути влаштовані так, як диктує логіка можливо більш повного запозичення компонентів зі світу комп'ютерів масового випуску і змінюватися за змінами цієї логіки;

2) дві суперкомп'ютерні революції дають нам два приклади конкретних рушійних сил, що породжують принципово нові технічні рішення. Так, найбільш масштабні зміни відбуваються через зростання кількості транзисторів на кристалі мікросхеми (перша суперкомп'ютерна революція). Не настільки масштабні, але також дуже важливі зміни відбуваються, коли у світі комп'ютерів масового випуску різко змінюються комунікаційні, мережеві технології (друга суперкомп'ютерна революція). На даний момент обидва ці фактори діють одночасно і взаємопов'язано, породжуючи «кумулятивний ефект». Це з неминучістю має призвести (і вже наводить) до третьої суперкомп'ютерної революції, яка обіцяє змінити вигляд суперкомп'ютера так сильно, як він ще не змінювався за всю історію сучасної інформатики. Спочатку оцінимо масштаб зростання кількості транзисторів на кристалі, який стався за час з першої суперкомп'ютерної революції до наших днів. Забігаючи вперед, зазначимо, що зміни в комунікаційних технологіях, які нам ще належить обговорити, самі є прямим наслідком зростання кількості транзисторів на кристалі. Наприкінці 60-х років XX ст., коли С. Крей ще тільки вигадував Сгау-1 — одну з супермашин «домногопроцесорної» епохи, у процесорі було близько 15 тис. транзисторів. Приблизно через 20 років

«добагатопроцесорна епоха» скінчилася, оскільки кількість транзисторів на кристалі, тобто. у процесорі, перевалило за мільйон.

Минуло ще 20 років. Сьогодні у сучасному складному та швидкому процесорі транзисторів вже мільярди. Транзисторів на кристалі стало більше, ніж на початку першої суперкомп'ютерної революції було в мікропроцесорах усієї обчислювальної системи. Вочевидь, всі передумови чергового технологічного перевороту — третьої суперкомп'ютерної революції. Починаючи розгляд такої ризикованої речі, як прогноз (нехай і короткостроковий) напрямів розвитку галузі, слід ще раз уточнити терміни. Саме поняття суперкомп'ютерних революцій у тому вигляді, як воно формулюється в тут, зовсім не претендує на повноту охоплення всіх ідей, підходів і технологій, що існують у суперкомп'ютерній галузі. Наприклад, вище ми говорили про другу суперкомп'ютерну революцію як про перехід до панування у галузі кластерних технологій. Це, однак, зовсім не скасовує того факту, що протягом усього «кластерного» десятиліття такими фірмами, як Сгау, інтенсивно та успішно розвивалися технології побудови суперкомп'ютерів на оригінальній схемотехнічній (і навіть елементній) базі. Це, у свою чергу, не скасовує того факту, що протягом згаданого проміжку часу саме кластери переважній мірі визначали цей стан галузі. При цьому ступінь кількісного переважання кластерної технології над технологіями Сгау повною мірою з'ясувався «заднім числом», на досвіді «кластерного» десятиліття. Приблизно так само слід ставитись до пропонованих тут міркувань про третю суперкомп'ютерну революцію.

Жодною мірою не претендую на вичерпну систематизацію (і навіть досить повний огляд) всього того, що відбуватиметься в галузі в найближчі роки. Йдеться лише про те, що напрями розвитку суперкомп'ютерних архітектур, про які йтиметься нижче, видаються назрілими, підготовленими об'єктивним ходом усієї попередньої історії галузі. Можливо, їм належить зіграти ту саму роль, яку зіграли раніше кластерні технології. Може статися і так, що тут «переглянуті» інші тенденції розвитку, за якими насправді майбутнє. Можливо (і навіть дуже ймовірно), що галузь піде шляхом, який прогнозується, але не

тільки по ньому. Конкретну, точну відповідь на ці питання ми дізнаємося років через 10, на порозі чергової — четвертої — суперкомп'ютерної революції, про яку поки що можна сказати лише те, що вона обов'язково буде. Поки що спробуємо описати контури третьої, не забуваючи, що це лише прогноз одного з напрямків розвитку.

Перший напрямок — докорінна зміна комунікаційних технологій. Зростання числа транзисторів на кристалі уможливило «стирання граней» між способами інтеграції функціональних пристроїв у межах материнської плати та способами інтеграції материнських плат у локальну мережу. Наслідки цих змін у комунікаційних технологіях самі собою досить революційні для того, щоб їх варто було розглянути уважно та докладно. Однак, як ми побачимо під час розгляду, третя суперкомп'ютерна революція до цих змін зовсім не зводиться. Другий напрямок третьої суперкомп'ютерної революції полягає у змінах організації самих обчислювачів. Фоннейманівський процесор практично перестав бути адекватною обчислювальним додаткам формою організації тієї кількості транзисторів, яка може бути розміщена на кристалі сьогодні. Між першим і другим напрямками чергової суперкомп'ютерної революції є як глибинний взаємозв'язок, так і істотні відмінності. З одного боку, перший напрямок є причиною, необхідною умовою другого. Хоч би якими були нові обчислювальні структури, що йдуть на зміну традиційним процесорам, побудова з них систем масового паралелізму навряд чи можлива без нових комунікаційних технологій, так що «обчислювальний» напрямок без «комунікаційного» багато в чому втрачає сенс. З іншого боку, перший, комунікаційнй напрямок, має самостійну цінність, навіть якщо розробка перспективних обчислювальних структур не передбачається, а йдеться про використання звичайних процесорів. При цьому загальні контури майбутніх змін у комунікаційних технологіях вже більш-менш очевидні. Питання про те, як саме виглядатиме нове комунікаційне середовище, не стоїть. Вже сьогодні це більш-менш ясно. З третього боку, про другий, обчислювальний напрям нічого подібного сказати не можна. Тут не зрозуміло практично нічого. Існує цілий ряд підходів до побудови альтернативних обчислювальних архітектур, і

величезна кількість невирішених питань. Ясно лише, що майбутнє — за новими архітектурами, але за якими саме - питання.

Магістраль обміну даними із процесором: шина чи мережа? Донедавна комунікаційні технології, які застосовувалися для зв'язку між функціональними пристроями в межах материнської плати (внутрішні), принципово відрізнялися від комунікаційних технологій, які застосовувалися для зв'язку між материнськими платами вузлів у межах обчислювального кластера (зовнішніх). Внутрішні комунікації здійснювалися з допомогою системи загальних шин, тоді як зовнішні комунікації використовували устаткування локальних мереж.

Загальна шина – це комунікаційна магістраль, здатна забезпечити дуже високі швидкості та дуже малі затримки під час передачі даних. Платою за високі показники ефективності є велика кількість провідників (десятки та сотні), які узгоджено змінюють свій стан на дуже високих частотах, причому за досить складними правилами. Це передбачає, у свою чергу, гранично жорсткі вимоги до якості фізичної реалізації: довжин з'єднань, кількості роз'ємів, якості розкладання провідників по шарах плати тощо. В результаті виведення внутрішньої комунікаційної магістралі за межі материнської плати з метою побудови єдиної комунікаційної магістралі, яка охоплює кілька материнських плат, стає або неможливим, або дуже дорогим. Саме тому для зв'язку материнських плат між собою застосовується обладнання локальних мереж, побудоване за іншими, ніж загальна шина, принципами. У мережному каналі кількість провідників зведено до мінімуму, правила, за якими ці провідники змінюють свій стан, гранично спрощені, вимоги до надійності знижені за рахунок контрольного підсумовування даних і повторення передачі збійних пакетів. Це дозволяє різко знизити вимоги до якості фізичної реалізації і як наслідок, використовувати кабельні з'єднання, тобто. виводити канал за межі материнської плати. Платою за таку адаптацію форми подання даних до комунікаційних потреб є величезні, порівняно із загальними шинами, затримки (латентність) передачі даних. Перетворення даних у форму мережевих пакетів відбувається багато в чому програмно, а в іншому - за допомогою порівняно повільного обладнання мережевої карти, і все це вимагає в десятки, в сотні

разів більшого часу, ніж акт спрацьовування загальної шини. Власне, у цій різниці між внутрішньою та зовнішньою комунікаційними магістралями і криється причина як відсутності загальної пам'яті в більшості сучасних суперкомп'ютерів, так і виключно високої ціни тих з них, у яких загальна пам'ять все ж таки передбачена. Адже саме висока латентність, органічно властива мережам, заважає організації на їх базі загальної пам'яті в межах суперкомп'ютера. У міру зростання робочих частот і розрядності процесорів вимоги до якості фізичної реалізації, які накладаються загальними шинами, ставали дедалі більш обтяжливими навіть для проектування та виготовлення материнських плат в умовах масового виробництва. У розробників материнських плат виникло бажання відмовитися від загальних шин, перевести внутрішні комунікації на мережеву технологію. Для того щоб не втратити при цьому продуктивності, необхідно було забезпечити кожну велику мікросхему, починаючи з процесора, «вбудованою мережевою картою» та «вбудованим мережевим процесором».

У міру зростання кількості транзисторів на кристалі така витрата частини кристала на комунікаційні потреби стала не просто можливою, а навіть не дуже обтяжливою. Сьогодні перехід на мережеві технології інтеграції материнських плат (Hypertransport [3,5], PCI Express [2,4]) відбувся. Технології комунікацій усередині материнської плати стали дуже схожими технології зовнішніх комунікацій. Зокрема технології внутрішніх комунікацій різко спростилися в частині фізичної реалізації, і стали допускати кабельні з'єднання на метрових відстанях.

Декілька тез про те, що це насправді означає для розробників суперкомп'ютерів.

1. Відтепер вузли, які об'єднуються в кластер, можуть бути пов'язані не набагато менш тісно, ніж функціональні пристрої в межах материнської плати. Зокрема, процесори кластеру можуть безпосередньо адресувати «чужу» пам'ять та «чужі» зовнішні пристрої. Відмінність у часі доступу до своєї та чужої пам'яті повністю не зникає, але скорочується на порядки порівняно із традиційними MPP-системами.

2. Наслідком такої уніфікації внутрішнього та зовнішнього комунікаційного обладнання стає розмивання межі вузла в кластері. Кластер, тобто. багатопроцесорний обчислювач, побудований зі стандартних системних блоків масового випуску, набуває всіх позитивних властивостей NUMA-сервера. При цьому і обсяг комунікаційного обладнання, і ціна такого рішення не більша, а менша, ніж у випадку традиційного кластеру.

3. В результаті зникає необхідність, яка існувала багато років, відмовлятися від загальної пам'яті для зниження вартості суперкомп'ютера — реалізація загальної пам'яті тепер практично нічого не варта, і причини для використання технологій паралельного програмування, спеціально розрахованих на її відсутність, зникають.

4. Заміна класу комунікаційного обладнання, що переважає в категорії доступних суперкомп'ютерів (NUMA замість мереж двосторонніх обмінів) — зміна сама по собі цілком революційна. Воно не може не вплинути на весь технологічний ланцюжок застосування суперкомп'ютерів, починаючи з розробки паралельних алгоритмів і закінчуючи розробкою трансляторів і бібліотек. Зміна ця в цілому благотворна - технології паралельного програмування для NUMA-систем набагато простіші та зручніші. Проте сам цей технологічний переворот жодною мірою не є обіцяною третьою суперкомп'ютерною революцією. Він є її важливою, проте досить малою частиною.

Слід зазначити, що не йдеться про «відмирання» локальних мереж як таких. Локальні мережі загального призначення, побудовані за різними технологіями, різної продуктивності і вартості, нікуди не дінуться, але перестануть бути переважним способом побудови суперкомп'ютерів за кластерною технологією.

Архітектура фон Неймана за нинішньої кількості транзисторів на кристалі перестала бути цілком адекватною обчислювальним додаткам.

Така кількість транзисторів, вірніше логічних елементів, для ефективного використання саме обчислювальних додатків потребує, як мінімум, доповнення і розширення фоннеймановських принципів, а можливо, і в принципово інших формах організації. З практичної точки зору нам важливо встановити

взаємозв'язок цього загального затвердження із правилом економії сил на розробку суперкомп'ютера. Справді, якщо можливість серйозного підвищення продуктивності процесора будь-яким способом, наприклад, шляхом переходу на альтернативні архітектури, існує, то чому ми не бачимо її матеріалізованої у вигляді чергової моделі Pentium? Чи можна говорити про якісь нові розробки обчислювальних архітектур у рамках суперкомп'ютерної галузі, якщо набагато потужніша галузь комплектуючих масового випуску цим шляхом не йде? Обидва питання насправді є провокаційними, тобто поставленими не цілком коректно. Почнемо з того, що ми констатували недостатню адекватність фоннейманівської архітектури саме обчислювальним програмам. Добре відомо, що програми обчислювального характеру становлять малу частку додатків комп'ютерів масового випуску, на потреби яких і орієнтуються переважно виробники мікропроцесорів. Частка ж саме суперкомп'ютерних установок у споживанні стандартних мікропроцесорів, які випускаються сьогодні, просто мізерна — частки відсотка. Розробники суперкомп'ютерів, таким чином, вирішують просто інші завдання, ніж розробники стандартних мікропроцесорів, і немає нічого неможливого в тому, що для цього в сьогоднішніх умовах знайдеться інший інструмент, який не застосовується розробниками стандартних мікропроцесорів. Більше того, саме твердження про те, що розробники комплектуючих для техніки масового випуску не шукають нових архітектур, не є вірним.

Раніше на масовому ринку з'явилися, як мінімум, два вироби, побудовані на нефонейманівських процесорах:

- Ігрова приставка Sony PlayStation III. Її «серце» — мікропроцесор IBM Cell [6], який не є ні фоннейманівським процесором, ні навіть традиційним багатоядерним процесором із симетричним доступом ядер до загальної пам'яті;

- Відеоадаптери NVIDIA, побудовані за технологією CUDA GPU [7]. Крім застосування за прямим призначенням — як відеоадаптер, ці плати можна використовувати як векторні обчислювальні установки загального призначення, тобто вони мають у своєму складі потужний нефоннейманівський обчислювач. Крім того, існує цілий ряд розробок нових процесорів, що

випускаються малими серіями, на які фірми-виробники потужних серверів мають цілком конкретні плани. Таким чином, ми окреслили напрямок розвитку обчислювальних архітектур, який ніяк не можна оминути своєю увагою, а саме — мікропроцесори нетрадиційної архітектури, які серійно випускаються промисловістю, спеціально призначені для обчислювальних додатків. Напрямок цей — важливий, але не єдиний. Більше того, можливо, у середньостроковій перспективі навіть не основний. Стратегічно набагато більш цікавим і важливим видається розробка обчислювальних структур, які реконфігуруються, спеціалізованих під завдання або клас завдань. Зростання неадекватності фоннеймановської архітектури розмірам кристала, у поєднанні зі згаданими поступами на комунікаційному напрямку, дають цьому досить старому підходу новий шанс. Таким чином, проглядається два напрями — огляд порівняно нових виробів, які не існували раніше, але сьогодні вже серійно випускаються, і розробка перспективного обчислювального обладнання з новими можливостями, наприклад, технологій програмованої логіки.

Література

1. *Burks A. W.* Preliminary discussion of the logical design of an electronic computing instrument / A.W. Burks, H.H. Goldstine, J. von Neumann. // The report of the Prinston Institute of Advanced Researches, 1946.
2. http7/www. intel. com. Сетевой ресурс.
3. http 7/www, amd. com. Сетевой ресурс.
4. PCI Express System Architecture / R.Budruk, D. Anderson, T.Shanley.
5. *Trodden* J. Hypertransport System Architecture / J. Trodden, D. Anderson.
6. http7/www. ibm. com. Сетевой ресурс.
7. http://developer. nvidia. com/cuda. Сетевой ресурс.

5. 2 *Альтернативні архітектури обчислювача*

В попередніх лекціях ми переконалися, що навіть незначні вкладення зусиль і коштів у

власні розробки комунікаційного устаткування можуть дати значні переваги в параметрах готового виробу. Але це ніщо порівняно з виготовленням — або навіть просто використанням обчислювача альтернативної архітектури.

Нехай ми маємо системний блок із материнською платою, у якій замість процесора Pentium чи Power PC – систолічний масив. Що ми робитимемо з цим комп'ютером? Де Linux чи Windows, де транслятори, як підключитися до мережі? "Масштаб лиха" очевидний і не піддається виміру. Хотілося б відразу зробити найважливіше зауваження і разом «перекинути міст» до наступного викладу. Звісно ж, очевидно, що ні про яку заміну фоннеймановского процесора обчислювачем альтернативної архітектури мови у середньостроковій перспективі не може бути. Говорити можна лише про додавання до системи, побудованої на базі пов'язаних комунікаційним середовищем фоннейманівських процесорів, певної кількості співпроцесорів, які мають альтернативну архітектуру. Причин тому дві. Обсяг сервісних, у найширшому сенсі цього слова, функцій традиційного процесора настільки величезний, що будь-яка спроба «швиденько повторити» все це на принципово новій основі є технологічною авантюрою. Але це не головне. Головне — у тому, що прикладні програмісти звикали до фоннейманівського програмування майже 70 років і не знають нічого іншого. Будь-яка спроба вирішення реального обчислювального завдання на альтернативній архітектурі може закінчитися частковим або повним провалом. Архітектура може виявитися непридатною — хай навіть для одного — єдиного завдання. З фоннейманівською архітектурою такого статися не може — за 70 років ми навчилися робити на ній усе, і навіть трохи більше. Отже, слід передбачити можливість відступу на добре підготовлені позиції. Будь-який новостворюваний суперкомп'ютер загального призначення повинен бути добре - за сучасними йому мірками - оснащений традиційною обчислювальною потужністю, хоча, по можливості, не тільки нею. У програміста має бути можливість, користуючись як усім системним сервісом, напрацьованим за десятиліття, так і власним алгоритмічним багажем, гнучко ділити роботу між традиційними процесорами та альтернативними обчислювачами, тобто суперкомп'ютер має бути комбінованим. Як ми знаємо з прикладу розгляду проблематики метакомп’ютингу, для гнучкого поділу роботи дрібними порціями потрібна дуже хороша комунікаційна система. Мало толку від спеціалізованого обчислювача, якщо читання вихідних даних та перетворення їх у придатну для обробки на ньому форму займає у універсального процесора вдесятеро більше часу, ніж саме обчислення. Саме можливість «завантажити» новостворені співпроцесори, нарівні з функціональними пристроями стандартних материнських плат, у новостворене комунікаційне середовище, швидке і низьколатентне, і робить розмову про співпроцесори актуальною саме сьогодні.

Як уже зазначалося до проблематики співпроцесорів з альтернативною архітектурою,

можна підходити з двох сторін: розглядати рішення, які промисловість комп'ютерних комплектуючих серійного випуску вже пропонує спеціально для додатків обчислювального характеру, або готова ось-ось запропонувати; намагатися будувати проблемно-орієнтовані обчислювальні структури на основі програмованої логіки, залучаючи до цього цікавого процесу прикладних програмістів.

Слід взяти до уваги наступне. Процес проектування спеціалізованих обчислювачів організується як єдність двох фаз - проектування апаратної частини і розробки програмного забезпечення (ПЗ). Витрати на створення та відпрацювання ПЗ складають 70 ... 75% загальної вартості обчислювального комплексу. При проектуванні апаратури спеціалізованого обчислювача особлива увага приділяється розробці пристроїв зв'язку з об'єктом (ПЗО) - блоків сполучення та перетворення сигналів з датчиків і виконавчих пристроїв (витрати - до 70% вартості розробки апаратури). Таким чином, функціональність спеціалізованого обчислювача досягається за допомогою високоефективного і надійного програмного забезпечення, а також розвиненої номенклатури ПЗО. Специфіка ПЗ полягає в необхідності реалізації багатозадачності обчислень у реальному часі.

Також важливим аспектом проектування спеціалізованих обчислювачів є розробка каналів зв'язку обчислювальних пристроїв, які входять до складу різних підсистем об'єкта, один з одним. При цьому доводиться вирішувати такі завдання, як вибір фізичної реалізації каналу зв'язку, формування протоколів мережного обміну, розробка принципів кодування інформації на логічному і фізичному рівні, розробка або вибір апаратури, що реалізує мережевий обмін та узгодження каналів зв'язку з обчислювальними пристроями (мережних адаптерів).

Далі проаналізуємо коротко основні відомі сьогодні універсальні не фоннеймановскі архітектури обчислювачів, а також доступні для придбання вироби на їх основі.

Універсальні не фоннеймановські архітектури

Цей напрямок розпочався в 2007р, коли вперше в історії високопродуктивних обчислень було розпочато будівництво найбільшого (десятки тисяч процесорних ядер) суперкомп'ютера з використанням не фоннеймановських прискорювачів. Йдеться про суперкомп'ютер RoadRunner, який IBM будувала на замовлення дослідницького центру в Лос-Аламосі, США [11]. Саме тоді спеціалізовані прискорювачі стали "раптом" дуже популярними серед розробників нових суперкомп'ютерів. Розгляд цього напряму був дуже актуальним з практичної точки зору, оскільки йшлося про вироби дійсно масового випуску, цілком доступних тим, хто будує суперкомп’ютери.

Векторний процесор

Векторний процесор — найстаріша, найвідоміша і найпростіша у застосуванні альтернативна архітектура. Розширенням такого роду забезпечувалися традиційні процесори від Cray-1 до сучасних версій Pentium і PowerPC. Розширення полягає у додаванні до традиційної системи команд набору додаткових - векторних - команд, таких, наприклад, як команда "скласти покомпонентно два вектори чисел". Чим довші оброблювані вектори, тим більший швидкісний виграш. Cray-1 "умів" додавати однією командою десятки пар чисел [1]. Математики давно звикли до цієї можливості та інтенсивно її використовують, іноді навіть не підозрюючи про це, оскільки сучасні транслятори успішно «векторизують» програми без сторонньої допомоги. За рахунок чого досягається виграш швидкодії при використанні векторного процесора? Дивне питання, подумає ви, звичайно, за рахунок одночасного виконання багатьох арифметичних операцій багатьма арифметичними пристроями. І буде неправі, точніше, не цілком праві.

Звичайно, паралельне виконання багатьох арифметичних операцій над багатьма парами чисел дає пропорційний швидкісний виграш, але джерел прискорення рахунку при застосуванні векторного процесору існує, як мінімум, ще два. Перший — в економії роботи з обчислення адрес обчислюваних чисел, пропорційного довжині вектора. Сучасні процесори часто витрачають час не стільки на обчислення, корисні з точки зору прикладного програміста, скільки на обчислення адрес тих величин, з якими потрібно ці обчислення виконати. Векторний процесор обчислює адреси операндів своїх команд не для кожної пари операндів, а один раз для вектора максимально допустимої довжини. Це важливе джерело економії. Принаймні, сам запис програми у векторній формі, явний або неявний, вже сприяє тому, щоб втрат такого роду було якнайменше. Але набагато важливіше друге джерело економії. Щоб паралельно виконувати багато арифметичних операцій над багатьма парами чисел, ці числа треба вміти паралельно діставати з пам'яті і паралельно ж у неї записувати, тобто, векторний обчислювач має бути оснащений «векторною пам'яттю». У сучасних процесорах загального призначення «векторної пам'яті» немає, і саме цим пояснюється дуже скромна довжина вектора в «векторних вставках». У Cray-1 можливість оперувати векторами набагато більшої довжини забезпечувалася наявністю спеціальних векторних регістрів, тобто окремої векторної пам'яті невеликого обсягу. Ключем до успіху векторизації програми вважалося максимальне використання так званого зачеплення регістрів, тобто, якомога триваліша обробка даних, які вже потрапили у векторні регістри, без звернення до системної пам'яті [1].

Слід зазначити, що реалізація пам'яті, здатної одночасно видавати або приймати велику кількість значень з різних адрес, дуже дорога і складна на схемному рівні. Задача трохи спрощується і стає розв'язною, якщо реалізується можливість одночасно видавати або приймати значення, розташовані в пам'яті поспіль. Для векторного обчислювача потрібне

саме це. Такий окремий випадок пам'яті, здатної одночасно видати або прийняти багато значень, називається розшарованою пам'яттю. Вона складається з декількох окремих пристроїв пам'яті, число яких зазвичай дорівнює степені двійки. Ці пристрої пам'яті, звані шарами, об'єднані таким чином, що в адресному просторі об'єднаної пам'яті клітинки шарів чергуються циклічно. При цьому будь-які N значень, розташованих в об'єднаній пам'яті поспіль, потрапляють у різні шари, якщо значення N не перевищує числа шарів. Саме розшарована пам'ять і застосовується у векторних обчислювачах як «векторна пам'ять». Ідеальний векторний обчислювач повинен мати пам'ять, порівнянну за розміром із системною пам'яттю універсального комп'ютера, повністю розшаровану, з числом шарів, рівним числу арифметичних пристроїв в обчислювачі (рис. 1).

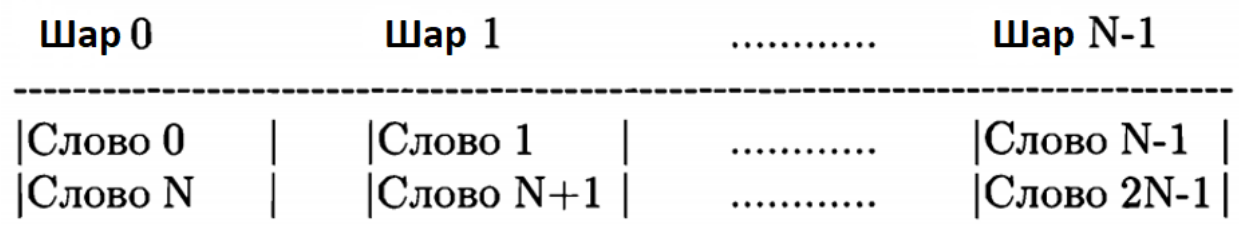


Рис. 1 Пам’ять із N шарів (кожний шар – окремий пристій пам’яті)

Як приклад серійно випускаємого і цілком доступного на ринку векторного обчислювача з архітектурою SIMD і глибоко розшарованою пам'яттю розглянемо відеоадаптер компанії NVIDIA, побудований за технологією CUDA GPU [2]. Ця техніка стала вперше доступна користувачам наприкінці 2006 року. Чому ми назвали відеоадаптер векторним обчислювачем і який тут зв'язок? Відеоадаптери сучасних комп'ютерів масового випуску зосереджують у собі величезну обчислювальну потужність, необхідну побудові реалістичних зображень з опису тривимірної сцени. Це є типовим обчислювально-ємним додатком. Оскільки додаток постійно використовується в процесі роботи комп'ютера і завжди хочеться, щоб він виконувався швидше, його намагаються реалізувати не програмно, на універсальному процесорі, а апаратно, на спеціальному «графічному процесорі» у складі відеоадаптера. Графічний процесор працює набагато швидше універсального, але система команд його настільки спеціалізована під графіку, що практично ні для чого іншого не підходить.

Бажання все ж таки використовувати обчислювальну потужність графічного процесора не тільки для формування картинки на екрані монітора, але і для загальної обчислювальної роботи, з'явилося у математиків не вчора. Періодично робилися спроби, іноді успішні, звести обчислювальну роботу до послідовності таких дій, які зустрічаються в «графиці», щоб доручити цю роботу графічному процесору. Зрештою, розробники відеоадаптерів пішли назустріч таким спробам, позбавивши ентузіастів графічних обчислень від утомливих

намагань «добування сиру з вареників у промислових масштабах». Було вирішено зробити графічний процесор набагато більш універсальним, узагальнити його архітектуру, щоб вона, ставши «програмованішою», годилася не тільки для графіки. В результаті вийшов GPGPU - графічний процесор з архітектурою загального вигляду, придатною для універсального програмування. Ахітектура отримала назву Cuda (Compute Unified Device Architecture, Уніфікована обчислювальна архітектура пристрою).

З погляду універсального процесора материнської плати, CUDA GPU це співпроцесор. У комунікаційну магістраль материнської плати він включається через «графічний» роз'єм PCI Express шириною 16, що має пікову пропускну здатність близько 4 Гб/с, прийом і передачу одночасно. Підключення векторного прискорювача до універсального процесора більш вузьким каналом навряд чи мало б великий сенс. Співпроцесор має власну пам'ять розміром у сотні мегабайт та програмований універсальним процесором канал DMA для передачі програм та даних між пам'яттю універсального процесора та пам'яттю співпроцесора.

У складі співпроцесора є 4, 12 або 16, залежно від моделі SIMD-процесорів. Кожен SIMD-процесор має 8 арифметичних пристроїв, кожен - зі своїм набором скалярних регістрів загального призначення, та 16 кб векторної пам'яті, що розшарована на 16 шарів. Арифметичні пристрої – 32-розрядні, використовують числа з плаваючою комою та стандартні режими округлення. Пам'ять, доступна для обміну даними з універсальним процесором, адресується всіма SIMD-процесорами, але не є розшарованою, і доступ до неї приблизно в 100 разів повільніший, ніж у розшаровану пам'ять SIMD-процесора. Для програмування розроблений спеціальний транслятор мови С, і досить нетривіальна програмістська модель, що не цілком безпосередньо відображається на апаратуру. Вона чудово документована [2], і розглядати її тут ми не будемо.

Найважливішим висновком із цього гранично короткого огляду є те, що тепер ми можемо вважати доведеною «конструктивну теорему існування». Завдання побудувати універсальний обчислювач, зовсім не схожий по архітектурі на процесори загального призначення, була не тільки поставлена промисловістю комп'ютерних виробів масового випуску, а й успішно вирішена. Головний аргумент проти переходу на нові архітектури, який полягає в тому, що «якби швидший процесор був можливий, він називався б Pentium», спростований експериментально. Альтернативні універсальні архітектури можливі, реально існують і їх слід наполегливо вивчати. І ще - найпотужніші сьогоднішні суперкомп’ютери реалізовані з використанням прискорювачів, в чому можна переконатися, уважно прочитавши попередню лекцію.

Процесор IBM Cell

Мікропроцесор IBM Cell цікавий для нас тим, що є виробом масового випуску: на його

базі побудована ігрова приставка Sony PlayStation III. Архітектура була розроблена спільно фахівцями Sony, Toshiba та IBM [3]. Мікропроцесор є гетерогенним багатопроцесорним обчислювачем — свого роду «суперкомп'ютер на кристалі». До його складу входить універсальний процесор PowerPC та вісім процесорних елементів. Всі ці дев'ять процесорних ядер об'єднані високошвидкісною загальною шиною, до якої також підключені два контролери зовнішньої пам'яті з піковою швидкістю передачі даних 12,8 Гб/с кожен, і два контролери введення-виводу зі швидкістю 38,4 Гб/с кожен. Робоча частота – близько 4 ГГц. Кожен процесорний елемент - це векторний процесор з набором із 128 регістрів розрядності 128, з чотирма арифметичними пристроями з плаваючою комою одинарної точності, і з чотирма арифметичними пристроями цілої арифметики. Крім того, до складу процесорного елемента входить 256 Кб локальної пам'яті. На загальну шину кристала процесорний елемент виходить через контролер DMA, здатний обмінюватися даними між локальною пам'яттю та загальною шиною.

Локальна пам'ять процесорного елемента не кешується, але працює зі швидкістю кеш. Кеш-пам'яті як такої у процесорному елементі просто немає. Зовнішня пам'ять, яка підключається до кристала, адресується універсальним процесором, але не процесорними елементами. Процесорні елементи можуть виконувати односторонні обміни між своєю локальною пам'яттю і зовнішньою пам'яттю, підключену до кристалу. Подивившись на цю архітектуру ще раз, не можна не вразитись її простоті. Це — класична MPP-система, з керуючою машиною, і вісьма обчислювальними вузлами, які пов'язані мережею односторонніх обмінів — тільки все це знаходиться всередині кристала. Щоправда, "обчислювальні вузли" оснащені векторними розширеннями та обсягом регістрової пам'яті, достатнім для "зачеплення операцій" у стилі Cray-1 [2].

Мультитредова архітектура

Дослідження у сфері мультитредової архітектури мають давню історію, і на їх результати покладаються серйозні надії. Мабуть, сьогодні цей напрям можна вважати головним у створенні універсальних обчислювальних архітектур, відмінних від традиційних. Детальний виклад проблематики мультитредових (multithreaded) архітектур нашою метою не є. Обмежимося оглядом основних ідей. Як відомо, процес виконання команд у процесорі включає роботу різних функціональних пристроїв. При обчисленні індексів використовуються цілі арифметичні пристрої, при виконанні арифметичних операцій з плаваючою комою - пристрої арифметики з плаваючою комою. Також відбуваються звернення до пам'яті, в середньому — у десятки та сотні разів повільніші, ніж обробка отриманих із пам'яті значень. В послідовних процесорах минулого майже кожен такий функціональний пристрій у складі процесора практично весь час простоював. Спочатку

обчислювався індекс, потім відбувалося звернення до пам’яті, потім — арифметична операція з плаваючою комою, тощо. Для більш повного завантаження функціональних пристроїв з часом стали застосовуватися спеціальні прийоми, в першу чергу - конвеєрне виконання команд з розбивкою їх на стадії, а також кеш-пам'ять. Це дозволило значно підвищити середнє завантаження функціональних пристроїв процесора, як наслідок — його швидкодію. Втім, значний дисбаланс між часом доступу в пам'ять і часом обробки даних зберігся, незважаючи на інтенсивне використання кеш-пам'яті.

Мультитредінг (multithreading) є спробою набагато радикальнішого вирішення цієї проблеми, ніж це намагалися робити раніше. Уявімо, що процесор виконує багато процесів у псевдопаралельному режимі, але перемикання процесора між такими процесами виконується апаратно та дуже швидко, в ідеалі – на кожному такті робочої частоти процесора, у тім рахунку – «всередині» окремих процесорних команд. Тоді, якщо серед тих, хто чекає черги на виконання процесів, є хоча б один, якому вже час виконувати арифметичну операцію з плаваючою комою — зайнятість відповідного пристрою на наступному такті забезпечена. Якщо деякий процес надовго «застряг» на зверненні до пам'яті — нічого страшного, ті функціональні пристрої, які він збирається використати, але не може, будуть використані іншими процесами. Підібравши відносну чисельність функціональних пристроїв відповідно до їхньої відносної швидкодії, можна забезпечити повне завантаження всього обладнання процесора без «вузьких місць».

Працездатність цієї схеми, представленої в гранично загальному вигляді, залежить від двох факторів: - реалізації в процесорі на апаратному рівні дуже швидкого і розумного планувальника процесів; — можливості розбиття написаної прикладним програмістом програми на дуже багато паралельних процесів, які взаємодіють у моделі SMP.

Йдеться не про десятки, а, швидше, про десятки тисяч процесів. Процесори, орієнтовані на цю технологію, розробляються і випускаються, в крайньому разі, дослідними партіями, багатьма серйозними фірмами, такими, як Sun і Cray [4,5]. Тим не менш, роботи з практичного застосування мультитредової архітектури розвивалися не дуже швидко. Проте останнім часом процес пішов. Багато елементів мультитредової архітектури використані в архітектурі CUDA. Зокрема, виключно високу (200 тактів) латентність звернення до загальної пам'яті обчислювача вдається приховати саме розглянутим щойно способом з використанням вбудованого мультитредового планувальника при розбитті додатка на багато сотень легковажних процесів. Реалізація мультитредової архітектури у складі виробу дійсно масового випуску дозволить у короткий термін вивести цю архітектуру на рівень значно ширшого практичного застосування, ніж раніше.

Прблемно-орієнтовані обчислювачі

Вище було наведено кілька прикладів більш менш універсальних обчислювальних архітектур, покликаних так чи інакше подолати недостатню адекватність традиційних процесорів додаткам обчислювального характеру. Однак, до цієї проблеми можна підійти і з іншого боку. Новий, не схожий на стандартний процесор метод організації транзисторів в обчислювальну систему не повинен бути універсальним. Теоретично цілком можна уявити спеціалізований обчислювач, який реалізує той чи інший типовий алгоритм безпосередньо на апаратному рівні. Відповідь на цілком природне питання про те, скільки це буде коштувати, як саме робитися, і скільки часу займе, доки відкладемо, втім, зовсім ненадовго. Спочатку постараємося зрозуміти, за рахунок чого в принципі такий обчислювач міг би працювати швидше, ніж програма, яка реалізує той самий алгоритм на універсальному процесорі. Для цього згадаємо одне з обговорюваних вище джерело прискорення векторного процесора порівняно з універсальним. Векторний процесор економить час, зокрема, за рахунок того, що не обчислює адреси для кожного з елементів оброблюваних векторів. Той факт, що оброблювані значення, з якими потрібно виконати деякі однотипні дії, розташовані в пам'яті підряд, безпосередньо врахований у схемі самого процесора, і програмі для векторного процесора вже немає потреби враховувати це, виконуючи цикл проходу по елементах масивів. Відповідні цикли «перемістилися» із програми до структури схеми. При цьому схема не виконує їх швидше, ніж програма на універсальному процесорі - вона просто влаштована так, що виконувати тут алгоритму стало нічого. Таким чином, векторний процесор є спеціалізованим обчислювачем. Він добре виконує ті програми, які можуть бути записані у векторній формі, і досить непотрібний для додатків, які так записати не можна в принципі.

Однак спеціалізація обчислювача під всілякі додатки, які допускають векторизацію - це спеціалізація досить слабка. З програми, написаної для традиційного процесора, під час переходу на векторний процесор «переміщається у структуру схеми» лише мала частина коду, лише цикли проходу по векторам. Наслідуючи далі цим шляхом, тобто перекладаючи в структуру схеми все більшу і більшу частину програмного коду, можна отримати схему, яка взагалі не потребує програми, і витрачає час тільки на корисні, з точки зору прикладного програміста, обчислення. Якщо ця схема ще й має внутрішній паралелізм, вона має шанс працювати в сотні разів швидше за програму для універсального процесора при тих же робочих частотах елементної бази. Поняття пікової та реально досяжної швидкодії для такої схеми збігаються. Правда, виконувати вона завжди буде рівно один алгоритм, той, який у неї «зашитий». "Перезашити" на іншу програму її не можна, оскільки програми в ній немає. Для іншого алгоритму доведеться збудувати іншу схему. У цьому підході можна побачити деяку аналогію з порівнянням швидкості виконання програми, написаної мовою, що

інтерпретується і компілюється. Справді, програма на класичному Бейсику може виконуватися у двох варіантах: в інтерпретованому та скомпільованому. У другому випадку компіляція виконується заздалегідь, що заощаджує час виконання у рази. У описуваному нами підході пропонується виконати «глибшу компіляцію»: не в команди універсального процесора, а структуру схеми. Іншими словами, усунути ще один рівень інтерпретації.

Цей підхід тільки здається «нетрадиційним». В дійсності, він старший, ніж саме поняття процесора. На зорі обчислювальної техніки майже всі установки, які сьогодні прийнято називати «прообразами сучасних комп'ютерів», були, насправді, машинами однієї задачі. Наприклад, машиною для обробки даних перепису населення, або чимось подібним. Відбувшийся ж у середині XX ст. перехід від таких спеціалізованих машин до універсальних процесорів являв собою абсолютно нетривіальне нововведення, яке нелегко було придумати, тобто «справді нетрадиційним» став універсальний процесор. Вже на сучасному етапі комп'ютерної історії періодично виникали такі завдання з обробки даних, які виправдовували побудову для них спеціальних обчислювальних пристроїв. Геофізикам 70-х років XX ст. напевно добре знайомі «Фур'є-процесори». Набагато ближчий більшості приклад - графічні процесори, про які ми вже говорили вище. Задовго до того, як зростання розміру кристала HBIC дозволило перетворити їх на універсальні, програмовані обчислювальні структури, на кшталт CUDA GPU, вже існували спеціалізовані, непридатні для обчислень загального вигляду, «графічні машини», оформлені як відеоадаптери, або ж інтегровані особливо потужні робочі станції. Ще 30 — 40 років тому цілком на спеціалізованих під графіку обчислювачах будувалися льотні тренажери - з генерацією в режимі реального часу необхідних зображень тодішні універсальні комп'ютери не могли впоратися. Таким чином, ми маємо, як мінімум, один приклад реального обчислювального додатка, який запропонованим шляхом вдалося прискорити в багато разів. Ця програма — побудова реалістичного зображення за описом тривимірної сцени. Тепер саме час згадати про питання ціни та часу реалізації, яке ми вже згадували вище, але відклали «на потім». Відеоадаптери випускаються тиражами, як мінімум, у сотні тисяч - можна постаратися, вкласти в проектування багато часу і сил. Льотні тренажери минулого будувалися роками, і навіть їхнє фізичне виготовлення, коли все спроектовано, займало не місяць і не два. При цьому працювали десятки, а то й сотні людей.

Технології прогамованої логіки

Протягом тривалого часу спеціалізовані обчислювачі на базі програмованої логіки та універсальні комп'ютери на стандартних процесорах існували «в паралельних світах». По ряду технічних причин останнім часом програмовану логіку почали вводити до складу суперкомп'ютерів загального призначення, як реконфігуровані співпроцесори, що

«допомагають» звичайним процесорам. В даний час такі суперкомп'ютери випускають Cray та Silicon Graphics [6,7].

Вказаною проблематикою доить серйозно займаються в нашому університеті на кафедрі обчислювальної техніки ФІОТ. Як було вказано в лекції 2 однією з особливостей кластеру НТУУ «КПІ» є наявність апаратного прискорювача операцій. У рамках третьої модернізації обчислювального комплексу НТУУ «КПІ» створено обчислювальний модуль на базі програмованої логіки ПЛІС (FPGA). Результати для вибраних алгоритмів показують доцільність використання цього типу обчислювальних модулів для виконання низки ресурсоємних задач.

Література

1 *Воеводин В. В.* Параллельные вычисления. — СПб. : БХВ- Петербург, 2002. — 608 с.

2 http://developer. nvidia. com/cuda. Мережевий ресурс.

3 Introduction To the Cell Multiprocessor / [J. A.Kahle, M.N.Day, H.P.Hofstee at all]. // IBM Journal of research and development. — Vol. 49. - № 4/5, 2005.

4 http7/www. crav. com. Мережевий ресурс.

5 http://www. sun, com. Мережевий ресурс.

6 http7/www. crav. com. Сетевой ресурс.

7 http://www. sgi. com. Сетевой ресурс.

6 Інформаційні технології Grid computing для виконання високопродуктивних обчислень.

6.1. Grid computing

Аналізуючи історію розвитку інформаційних технологій та сучасні тенденції можна зробити висновок, що еволюційний виток ІТ, який почався разом з епохою великих обчислювальних машин (ЕОМ) понад п'ятдесят років тому, замкнувся – разом з хмарами ми повернулися до централізації ресурсів, але на цей раз не на рівні мейнфреймів, а на новому технологічному рівні.

Виступаючи на конференції, присвяченій проблемам сучасних процесорів, професор Массачусетського технологічного інституту Ананд Агарвал сказав: "Процесор – це транзистор сучасності. Новий рівень відрізняється тим, що тут також збираються мейнфрейми, але віртуальні, і не з окремих транзисторів, як півстоліття тому, а з цілих процесорів, або цілком з комп'ютерів. На зорі ІТ численні компанії та організації створювали власні комп'ютери з дискретних компонентів, монтуючи їх на саморобних друкованих платах - кожна

організація робила свою машину, і ні про яку стандартизацію чи уніфікацію мови не могло бути. Зараз ситуація повторюється – точно так саме: з серверів, комп'ютерів, різноманітного мережевого обладнання збираються створюються складні і дуже потужні системи. Одночасно спостерігається та ж сама технологічна роз'єднаність і відсутність уніфікації: Microsoft, Google, IBM, Aptana, Heroku, Rackspace, Ning, Salesforce будують глобальні мейнфрейми, а хтось під власні потреби створює приватні складні структури. Залишається припустити, що попереду винахід «інтегральної схеми і мікропроцесора".

Інтернет, WWW - World Wide Web, Grid і хмарні обчислення - пов'язані між собою, але різні технології:

• ***Інтернет*** - це глобальна мережа, що об'єднує безліч комп'ютерів і локальних (порівняно невеликих) мереж і дозволяє їм взаємодіяти один з одним.

• ***Веб*** (паутина) - це спосіб доступу до інформації, яка знаходиться на віддаленому, але включеному в Інтернет комп'ютері.

• ***Web-служби (Web Services)*** - це віддалені сервісні об'єкти, які реалізують за запитом користувача деяку функціональність.

Історично першими з'явилися великі комп'ютери, які в західній термінології називають ***"мейнфрейми"***, у них відсутні обмеження на обсяг обладнання, тому їх називають ще великими ЕОМ. Це відбулося в 1964 році, коли компанія IBM створила перший комп'ютер сімейства IBM-360, які були багато років присутні на ринку ЕОМ. Слід зауважити, що в той час в Україні була створена Інститутом Кібернетики ЕОМ "Дніпро-2" - повний функціональний аналог IBM-360: 64-розрядна ЕОМ з 128-розрядною "плаваючою" арифметикою, оперативною пам'яттю до 65536 байт і повним набором периферійних пристроїв (пристрій паралельного друку АЦПУ-128, пристрій послідовного друку, пристрої вводу і виводу інформації на перфострічку, пристрої вводу та виводу на перфокарту, зовнішня пам'ять на магнітній стрічці в складі 8-ми накопичувачів, екранний пульт: алфавітно-цифровий та графічний та ін.). Ця ЕОМ ("Дніпро-2") серійно вироблялась в Києві до 1971р.

СуперЕОМ - це продовження великих ЕОМ в бік збільшення числа АЛУ, процесорів, цілих машин, об'єднаних в одну систему для виконання паралельних обчислень в рамках однієї задачі. Такі ЕОМ включають сотні і тисячі процесорів, розташованих локально, так що ефективність визначається простим законом Амдала.

Можливості централізованого обчислення обмежені, оскільки такі системи за своєю природою обмежені в обладнанні, тому для великих задач більший інтерес представляють розподілені системи, сумарна потужність яких за визначенням перевершує потужність будь-якої централізованої системи.

Коротко розглянемо засоби для організації таких разподілених систем.

Інтернет

Інтернет є єдиним інформаційним простором, в якому можна побудувати різні конструкції - сайти, хости, сервери і т.д. Історія Інтернету почалася з 1958 року, коли в відповідь на запуск першого супутника, США створили організацію під назвою ARPA (Advanced Research Projects Agency). До 68-го

року в ARPA і в інших організаціях велася робота по з'єднанню комп'ютерів у мережі. В 67-му році пройшов симпозіум трьох незалежних розробників комп'ютерних мереж: ARPA, NPL, RAND. В 1968-м році була побудована перша мережа, заснована на сучасних принципах Інтернету.
Вона складалася з 4-х комп'ютерів. Слід зазначити, що тоді ж (в 1968р) на заводі обчислювальних машин в м.Києві (вул. Велика Окружна 4) була створена автоматизована система управління АСУ підприємства в складі 4-х ЕОМ «Дніпро 2», які були розміщені в різних приміщеннях заводу і об'єднані швидкісною локальною мережею.

WWW - є спроба організувати всю інформацію в Internet, а також будь-яку локальну інформацію за вашим вибором як набір гіпертекстових документів. Ви переміщується по мережі, переходячи від одного документа до іншого по посиланнях. Всі ці документи написані на спеціально розробленій для цього мові, що називається HyperText Markup Language. World Wide Web, Всесвітня павутина, WWW, Web, Інтернет, - це все назви одного й того ж сервісу, який був придуманий в 1991 році і використовує протокол HTTP для передачі гіпертекстових документів та інших файлів від Веб- сервера до клієнтів. Головна відмінність WWW від інших інструментів для роботи з Internet полягає в тому, що WWW дозволяє працювати практично з усіма доступними зараз на комп'ютері видами документів: це можуть бути текстові файли, ілюстрації, звукові і відеоролики і т.д. Принцип роботи WWW наступний. Користувач запускає у себе програму, яка розуміє протокол HTTP і спеціальну мову, на якій створюється вміст WWW. Ця програма називається «програмою перегляду HTML-сторінки», або по-англійськи - browser (браузер). При цьому HTML (Hyper Text Markup Language) - це «мова розмітки гіпертексту», або мова, за допомогою якої створюється гіпертекст. Далі користувач набирає адресу www сервера. Браузер звертається до сервера з проханням віддати документ, розташований за цією адресою. Сервер віддає документ. Браузер отримує документ, обробляє його і, якщо в ньому є картинки, також просить сервер віддати йому їх, як і інші матеріали документу. Цей документ прийнято називати сторінкою, або WEB (Веб) -сторінкою, або HTML- сторінкою. Після цього браузер обробляє всі прийняті дані і показує готову сторінку на Вашому екрані. Деякі елементи сторінки (тест, картинки, кнопки) можуть бути посиланнями. Якщо Ви натиснете на них, то Ваш браузер пошле запит серверу, щоб попросити у нього документ.

Таким чином, Ви можете пересуватися від документа до документа, від сервера до сервера, що перетворює весь Інтернет в одну гігантську Мережу, яка пов'язує документи і сервери один з одним нитками гіперпосилань.

WWW - система в цілому складається з наступних компонент:

- Мова гіпертекстової розмітки HTML
- Протокол передачі гіпертексту HTTP
- Специфікацій на типи даних в Internet (Internet Media Types)
- Системи WWW-адресації (URL, URN, URI etc.).

Абревіатура URL розшифровується як Uniform Resource Locator, що можна перекласти, як "єдиний покажчик на ресурс". Практично, це адреса документа.

Що таке Web-сервіс? Web-сервіс - це серверний об'єкт, який реалізує певний елемент функціональності, з яким можуть взаємодіяти віддалені програми по протоколу HTTP за допомогою повідомлень на мові XML.

Технологія Web Services призначена для створення розподілених додатків, що функціонують в гетерогенному (неоднорідною) середовищі Інтернет , компоненти яких взаємодіють на базі стандартних Web - протоколів.

Грід

Під англійським терміном GRID (решітка) розуміється сукупність просторово розподілених обчислювальних вузлів, пов'язаних деякою мережею для обміну даними. Надалі замість GRID будемо використовувати слово Грід. Web Service дозволяє клієнту виконати на обладнанні власника ресурсу деяку функцію зі списку, складеного власником цього обладнання. Грід принципово відрізняється від Web Services.

Грід – це метод використання глобальних процесорних потужностей і систем зберігання інформації (дискові системи великої місткості) без їх фізичного переміщення в просторі. Грід включає як ресурси і все напрацьоване в WebServices. Протоколи WebServices (WSDL, SOAP, UDDI), засоби адресації в розширеному варіанті є основними протоколами Грід.

Грід характеризується наступним чином:

• Географія ресурсів глобальна

• Ресурси неоднорідні, відрізняються по апаратурі, операційним системам, протоколам функціонування і зв'язку

• Склад ресурсів змінний, в загальному випадку непередбачуваний

• Елементи середовища можуть бути ворожими, тобто можуть пошкодити або захопити закриті дані.

Існують наступні два основні різновиди Грід:

• Комунальний Грід. Такий Грід дозволяє надавати на запит клієнта відповідний ресурс.

• Метакомпьютер. В цьому випадку об'єднуються ресурси ряду корпорацій, можливо - весь Інтернет для вирішення загальнолюдських завдань. Структура Грід показана на рис.6.1. Тут клієнт спочатку звертається до реєстру ресурсів MDS, щоб отримати по заданих ним ознакам відомості про наявність потрібного ресурсу і місце його розташування. Клієнт по одній з отриманих адрес звертається до ресурсу, щоб отримати інтерфейс ресурсу. В інтерфейсі вказується спосіб завдання ресурсу необхідної роботи. Нарешті, клієнт передає ресурсу завдання і через деякий час отримує результат.

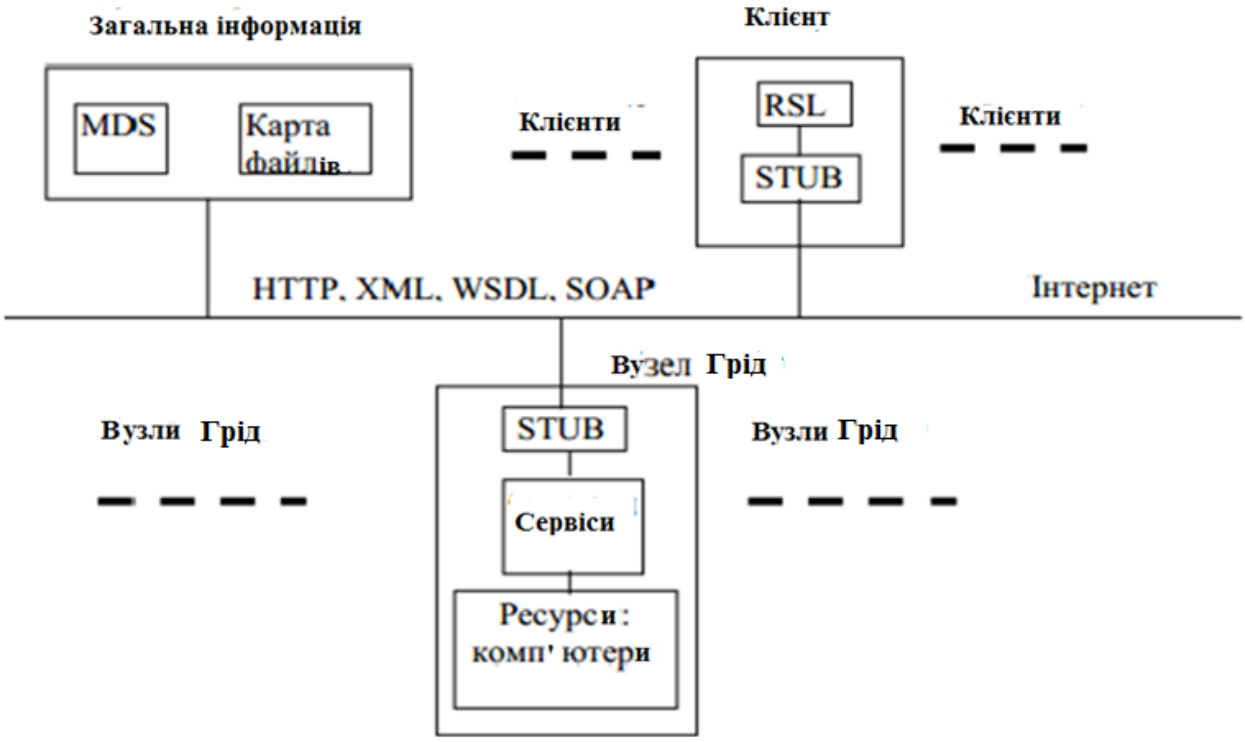


Рис. 6.1. Структура і функціонування Грід

Тут RSL (Resource Specification Language) - мова для опису завдання, яке потрібно виконати: програма, дані, де все це розміщено, куди послати результат, вимоги до ресурсів та ін.).

Отже, **Ґрід** — це географічно розподілена інфраструктура, яка об'єднує множину різних типів обчислювальних ресурсів , доступ до яких користувач може отримати з будь-якої точки, незалежно від місця їх розміщення. Ґрід є формою розподілених обчислень, в якому багато комп'ютерів об'єднані в один потужний віртуальний комп'ютер, і які працюють разом для виконання трудомістких завдань.

Ґрід-обчислення з'єднує комп'ютери з багатьох адміністративних доменів для досягнення певних цілей, Одніє з основних стратегій ґрід-обчислень є використання проміжного програмного забезпечення (ППЗ), яке може адаптуватися під завдання, що розв'язується в одному віртуальному домені, для того щоб розподілити програму серед декількох комп'ютерів, іноді навіть серед тисяч. Основним проміжним ПЗ є Globus Toolkit, gLite, UNICORE, EMI (Europen middleware initiative).

Координація додатків на Грід- системах може бути складним завданням, особливо коли координують потоки інформації через розподілені обчислювальні ресурси.

Грід-обчислення покладається на цілі комп'ютери (з повною комплектацією), під'єднані до комп'ютерної мережі (приватної або публічної) звичайним мережевим інтерфейсом, в той час як звичайний суперкомп'ютер містить безліч процесорів, підключених до локальної високошвидкісної шини. Також є певні відмінності у програмуванні та устаткуванні. Писати програми, які працюють у середовищі суперкомп'ютера, що може мати унікальну операційну систему, може бути дорого і складно. Якщо проблема може бути

адекватно розпаралелена, тонкий шар ґрід-інфраструктури може дозволити звичайній програмі запуститись на декількох машинах.

Термін «грід-обчислення» з'явився на початку 1990-х років, як метафора, що демонструє можливість простого доступу до обчислювальних ресурсів як і до електричної мережі (англ. Power grid), у збірнику під редакцією Яна Фостера і Карла Кессельмана "The Grid: Blueprint for a new computing infrastructure ".

Використання вільного часу процесорів і добровільного комп'ютингу стало популярним в кінці 1990-х років після запуску проектів добровільних обчислень GIMPS в 1996 році, distributed.net в 1997 році і SETI @ home в 1999 році. Ці перші проекти добровільного комп'ютингу використовували потужності приєднаних до мережі комп'ютерів звичайних користувачів для вирішення дослідницьких завдань, що вимагають великих обчислювальних потужностей.

Ідеї Грід-системи (включаючи ідеї з областей розподілених обчислень, об'єктно-орієнтованого програмування, використання комп'ютерних кластерів, веб-сервісів та ін.) були зібрані і об'єднані Іеном Фостером, Карлом Кессельманом і Стівом Тікі, яких часто називають батьками Грід-технології. Вони почали створення набору інструментів для Грід-комп'ютингу Globus Toolkit, який включає не тільки інструменти менеджменту обчислень, але й інструменти управління ресурсами зберігання даних, забезпечення безпеки доступу до даних і до самого Грід, моніторингу використання і пересування даних, а також інструментарій для розробки додаткових Грід-сервісів. В даний час цей набір інструментарію є де-факто стандартом для побудови інфраструктури на базі технології Грід, хоча на ринку існує безліч інших інструментаріїв для Грід-систем.

Спочатку, Грід - обчислення були орієнтовані на рішення наукових завдань за допомогою суперкомп'ютерних систем. Нині Грід застосовується для науково-дослідницьких завдань, вирішення яких вимагає об'єднання декількох суперкомп'ютерних платформ. З іншого боку, хмарні обчислення орієнтовані не на вирішення окремих завдань, а на перманентне надання певних сервісів кінцевим користувачам. Вони забезпечують динамічний розподіл фізичних ресурсів для задоволення змінного завантаження таких сервісів.

Також потрібно відмітити різну взаємодію з постачальниками ресурсів. Грід -обчислення грунтуються на понятті віртуальних організацій, що включають декілька різних окремих організацій з чіткими правилами взаємодії між ними і чіткими політиками надання програмно-апаратних ресурсів, в той час як хмарні обчислення забезпечують можливість будь-якій компанії використовувати хмарні сервіси для вирішення власних завдань, оплачуючи тільки ті ресурси, які потрібні. Грід -платформи надають базу для розгортання обчислювальної інфраструктури, в той час як хмарні обчислення надають інтегрований підхід на усіх рівнях для надання інформаційних ресурсів: інфраструктура як послуга, платформа як послуга, програмне забезпечення як послуга.

Архітектура Грід

Open GRID Services Architecture (OGSA) направлена на стандартизацію

адресації (для сумісності) за допомогою визначення основи структури додатку Грід. По суті стандарт OGSA визначає сервіси Грід, їх можливості і те, на яких технологіях вони засновані. Проте OGSA не розрізняє особливостей технічної сторони специфікації; метою є визначення – що є системою Грід. OGSA називають архітектурою, оскільки вона направлена на побудову і установку інтерфейсів, з яких можуть бути побудовані системи, засновані на відкритих стандартах WSDL.

Архітектура Грід визначає системні компоненти, цілі і функції цих компонент і відображає способи взаємодії компонент один з одним. Архітектура Грід є архітектурою взаємодіючих протоколів, сервісів і інтерфейсів, що визначають базові механізми, за допомогою яких користувачі встановлюють з'єднання з Грід -системою, спільно використовують обчислювальні ресурси для вирішення різного роду завдань.

Архітектура протоколів Грід розділена на рівні (рис.6.2), компоненти кожного з них можуть використовувати можливості компонент будь-якого з розташованих нижче рівнів. В цілому ця архітектура задає вимоги для основних компонент технології (протоколів, сервісів, прикладних інтерфейсів і засобів розробки ПО), не надаючи строгий набір специфікацій, залишаючи можливість їх розвитку в рамках прийнятої концепції.

Рис.6.2 Рівні архітектури протоколів Грід

Інфраструктура Грід заснована на наданні ресурсів в загальне користування, з одного боку, і на використанні публічно доступних ресурсів, з іншого. У цьому плані ключове поняття інфраструктури Грід – віртуальна організація, в якій кооперуються як споживачі, так і власники ресурсів. Мотиви кооперації можуть бути різними. У існуючих Грід - системах віртуальна організація є об'єднанням (колаборацію) фахівців з деякої прикладної області, які об'єднуються для досягнення загальної мети.

Грід - система є середовищем колективного комп’ютингу, в якій кожен ресурс має власника, а доступ до ресурсів відкритий в режимі, що розділяється за часом і по простору, безлічі користувачів, які входять до віртуальної організації. Віртуальна організація може утворюватися динамічно і мати обмежений час існування.

Ефективний розподіл ресурсів і їх координація є основними завданнями системи Грід і для їх вирішення використовується планувальник (брокер ресурсів). Користуючись інформацією про стан Грід - системи, планувальник визначає найбільш відповідні ресурси для кожного конкретного завдання і резервує їх для її виконання. Під час виконання завдання планувальник може запитати додаткові ресурси, або звільнити надмірні. Після завершення завдання всі відведені для нього обчислювальні ресурси звільняються, а ресурси пам'яті можуть бути використані для зберігання результатів роботи.

Важливою властивістю систем Грід є те, що користувачеві не потрібно знати про фізичне розташування ресурсів, відведених його завданню. Вся робота по управлінню, перерозподілу і оптимізації використання ресурсів лягає на планувальник і виконується непомітно для користувача. Для користувача створюється ілюзія роботи в єдиному інформаційному просторі, що володіє величезними обчислювальними потужностями і об'ємом пам'яті.

Грід є найбільш складним інформаційним середовищем, яке коли-небудь створювалося людиною. Для системи такої складності дуже важлива проблема забезпечення надійного функціонування і відновлення при збоях. Людина не здатна устежити за станом тисяч різних ресурсів, що входять в Грід - систему, і з цієї причини завдання контролю над помилками покладається на систему моніторингу, яка стежить за станом окремих ресурсів. Дані про стан заносяться в інформаційні ресурси, звідки вони можуть бути прочитані планувальником і іншими сервісами, що дозволяє мати достовірну інформацію, яка постійно оновлюється, про стан ресурсів.

У Грід - системах використовується складна система виявлення і класифікації помилок. Якщо помилка відбулася з вини завдання, то завдання буде зупинено, а відповідна діагностика направлена її власникові (користувачеві). Якщо причиною збою послужив ресурс, то планувальник проведе перерозподіл ресурсів для даного завдання і перезапустить його.

Збої ресурсів є не єдиною причиною відмов в Грід - системах. Через величезну кількість завдань і постійно змінної складної конфігурації системи важливо своєчасно визначати переобтяжені і вільні ресурси, проводячи перерозподіл навантаження між ними. Переобтяжений мережевий ресурс може стати причиною відмови інших ресурсів. Планувальник, використовуючи систему моніторингу, постійно стежить за станом ресурсів і автоматично приймає необхідні заходи для запобігання перевантаженням і простою ресурсів.

У розподіленому середовищі, яким є Грід - система, життєво важливою властивістю є відсутність так званої єдиної точки збоїв. Це означає, що відмова будь-якого ресурсу не повинна приводити до збою в роботі всієї системи. Саме тому планувальник, система моніторингу і інші сервіси Грід - системи розподілені і продубльовані. Не дивлячись на всю складність, архітектура Грід розроблялася з метою забезпечити максимальну якість сервісу для користувачів. У Грід - системах використовуються сучасні технології передачі даних, забезпечення безпеки і відмовостійкості.

Базовий рівень

Базовий рівень (Fabric Layer) архітектури Грід описує служби, що безпосередньо працюють з ресурсами. Ресурс є одним з основних понять архітектури Грід.

Можна виділити декілька основних типів ресурсів (рис.6.3):

· обчислювальні ресурси;
· ресурси зберігання даних;
· інформаційні ресурси, каталоги;
· мережні ресурси.

Рис. 6.3 Ресурси Грід

Обчислювальні ресурси надають користувачеві Грід -системи (точніше кажучи, завданню користувача) процесорні потужності. Обчислювальними ресурсами можуть бути як кластери, так і окремі робочі станції. При всій різноманітності архітектури будь-яка обчислювальна система може розглядатися як потенційний обчислювальний ресурс Грід -системи.

Необхідною умовою для цього є наявність спеціального програмного забезпечення, так званого ПЗ проміжного рівня (middleware), що реалізує стандартний зовнішній інтерфейс з ресурсом і дозволяє зробити ресурс доступним для Грід - системи. Основною характеристикою обчислювального ресурсу є продуктивність.

Ресурси пам'яті є простором для зберігання даних. Для доступу до ресурсів пам'яті також використовується програмне забезпечення проміжного рівня, що реалізує уніфікований інтерфейс управління і передачі даних. Основною характеристикою ресурсу пам'яті є його об'єм.

Інформаційні ресурси і каталоги є особливим видом ресурсів пам'яті. Вони служать для зберігання і надання метаданих і інформації про інші ресурси Грід - системи. Інформаційні ресурси дозволяють структуровано зберігати величезний об'єм інформації про поточний стан Грід - системи і ефективно виконувати завдання пошуку.

Мережевий ресурс є сполучною ланкою між розподіленими ресурсами Грід - системи. Основною характеристикою мережевого ресурсу є швидкість передачі даних. Географічно розподілені системи на основі даної технології здатні об'єднувати тисячі ресурсів різного типу, незалежно від їх географічного положення.

Рівень зв'язку

Рівень зв'язку (Connectivity Layer) визначає комунікаційні протоколи і протоколи аутентифікації.

Комунікаційні протоколи забезпечують обмін даними між компонентами базового рівня.

Протоколи аутентифікації, ґрунтуючись на комунікаційних протоколах, надають криптографічні механізми для ідентифікації і перевірки достовірності користувачів і ресурсів. Протоколи рівня зв'язку повинні забезпечувати надійний транспорт імаршрутизацію повідомлень, а також привласнення імен об'єктам мережі. Недивлячись на існуючі альтернативи, зараз протоколи рівня зв'язку в Грід-системах припускають використання тільки стека протоколів TCP/IP, зокрема: на мережевому рівні – IP і ICMP, транспортному рівні – TCP, UDP, на прикладному рівні – HTTP, FTP, DNS, RSVP. Враховуючи бурхливий розвиток мережевих технологій, в майбутньому рівень зв'язку, можливо, залежатиме і від інших протоколів.

Для забезпечення надійного транспорту повідомлень в Грід-системі повинні використовуватися рішення, які передбачають гнучкий підхід до безпеки комунікацій. В даний час ці рішення ґрунтуються як на існуючих стандартах безпеки, спочатку розроблених для Інтернет (SSL, TLS), так і на нових розробках.

Ресурсний рівень

Ресурсний рівень (Resource Layer) побудований над протоколами комунікації і аутентифікації рівня зв'язку архітектури Грід. Ресурсний рівень реалізує протоколи, які забезпечують виконання наступних функцій:
- узгодження політик безпеки використання ресурсу;
- процедура ініціації ресурсу;
- моніторинг стану ресурсу;
- контроль над ресурсом;
- облік використання ресурсу.

Протоколи цього рівня спираються на функції базового рівня для доступу і контролю над локальними ресурсами. На ресурсному рівні протоколи взаємодіють з ресурсами, використовуючи уніфікований інтерфейс і не розрізняючи архітектурні особливості конкретного ресурсу.

Розрізняють два основні класи протоколів ресурсного рівня:
· *інформаційні протоколи*, які отримують інформацію про структуру і стан ресурсу, наприклад, про його конфігурацію, поточне завантаження, політику використання;
· *протоколи управління*, які використовуються для узгодження доступу до ресурсів, які розділяються, визначаючи вимоги і допустимі дії по відношенню до ресурсу (наприклад, підтримка резервування, можливість створення процесів, доступ до даних). Протоколи управління повинні перевіряти відповідність запрошуваних дій політиці розділення ресурсу, включаючи облік і можливу оплату. Вони можуть підтримувати функції моніторингу статусу і управління операціями.

Список вимог до функціональності протоколів ресурсного рівня близький до списку для базового рівня архітектури Грід. Додалася лише вимога єдиної семантики для різних операцій з підтримкою системи оповіщення про помилки.

Стандарти побудови Грід

Термін «грид» був введений в використання Яном Фостером на початку 1998 року публікацією книги «Грід. Нова інфраструктура обчислень»: *Грід – це система, яка координує розподілені ресурси за допомогою стандартних, відкритих, універсальних протоколів і інтерфейсів для забезпечення нетривіальної якості обслуговування (QoS – Quality of Service).*

Хотя за останнє десятиліття базова ідея грід не зазнала суттєвих змін, всеохватного визначення грід не існує і досі.

Грід є технологією забезпечення гнучкого, безпечного і скоординованого загального доступу до ресурсів. У термінології Грід сукупність людей і організацій, які спільно розв'язують ту чи іншу загальну задачу і надають у користування один одному свої ресурси, називають *віртуальною організацією.*

Є два основні критерії, що виділяють Грід -системи серед інших систем, які забезпечують доступ до ресурсів, що розділяються:

· Грід - система координує розрізнені ресурси. Ресурси не мають загального центру управління, а Грід -система займається координацією їх використання, наприклад, балансуванням навантаження. Тому проста система управління ресурсами кластера не є системою Грід, оскільки здійснює централізоване управління всіма вузлами даного кластера, маючи до них повний доступ.

· Грід - система будується на базі стандартних і відкритих протоколів, сервісів та інтерфейсів. Не маючи стандартних протоколів, неможливо легко і швидко підключати нові ресурси в Грід -систему, розробляти новий вигляд сервісів і так далі.

Ключовим моментом в розробці Грід - додатків являється стандартизація, яка позволяє організовувати пошук, використання, розміщення та моніторинг різних компонентів, складаючих єдину віртуальну систему, навіть якщо вони предоставляются різними постачальниками послуг, чи керуються різними організаціями.

Найбільш важливим стандартом, покликаним визначити загальну, стандартну і відкриту архітектуру Грід, є стандарт Open Grid Services Architecture (OGSA). OGSA є сервіс-орієнтованою архітектурою, в якій специфікується набір розподілених обчислювальних патернів, що реалізовуються з використанням Web-сервісів. Стандарт призначається для визначення всіх основних сервісів, які можуть використовуватися в додатках e-business або e-science, включаючи управління роботами і ресурсами, комунікації і безпеку.

Стандарти, які використовуються для побудови архітектури Грід

Існує декілька стандартів, використовуваних для побудови архітектури Грід. Ці стандарти утворюють базові блоки, які дозволяють посилати запити додаткам і базам даних, а також дозволяють розгорнути програмне забезпечення, що дозволяє спростити управління бізнес - процесом.

До стандартів Грід і зв'язаних з ними стандартів слід віднести:

- Комунікації «програма - програма» (SOAP, WSDL, UDDI); (SOAP (Simple Object Access Protocol)- простий протокол для обміну ; WSDL (Web Services Definition Language)- опис інтерфейсу взаємодії компонентів розподіленої системи (мова описів веб-сервісів); UDDI (Universal Description, Discovery, and Integration)-стандартний протокол опису веб-сервісів і їх пошуку).
- Сумісне використання даних (мова XML).
- Передача повідомлень(SOAP, WS - Addressing, MTOM (для додатків));
- Надійна передача повідомлень (WS - Reliable Messaging);
- Управління робочим процесом (WS - Management);
- Управління транзакціями(WS - Coordination, WS - AtomicTransaction, WS - Business - Activity);
- Розподіл ресурсів (WS - RF або система ресурсних Web – сервісів);
- Забезпечення безпеки (WS - Security, WS - SecureConversation, WS - Trust, WS - Federation, Система безпечних зв'язків Kerberos для Web - сервісів;
- Обробка метаданих(WSDL, UDDI, WS - Policy).
- Orchestration (стандарти, використовувані для абстрагування бізнес - процесів від логіки додатків і джерел даних і для встановлення правив, які дозволяють бізнес - процесам взаємодіяти між собою);
- Верхній рівень управління бізнес - процесом (інженерна мова бізнес - процесу для Web сервісів - BPEL4WS)
- Події, що запускають бізнес - процес (WS - Notification).

Горизонтальна і вертикальна інфраструктура програмного забезпечення, яка необхідна для забезпечення безпеки, пошти, обміну повідомленнями, робочого потоку, колаборації, обмінів програма - програма, а також середовища сумісного використання даних може бути знайдена в інфраструктурних пропозиціях таких компаній як IBM (WebSphere), Microsoft (.NET), BEA (WebLogic) і Sun (ONE). Web - сервіси і XML реалізації містяться в пропозиціях інших постачальників.

Сервіс-орієнтована архітектура

Фундаментальною концепцією OGSA (*Open Grid Services Architecture*) є те, що сервісна архітектура складається з компонентів служб Грід, які працюють як спеціальні Web - сервіси, що надають ряд інтерфейсів, які відповідають спеціальним вимогам. .

SOA (Service Oriented Architecture – архітектура, орієнтована на сервіс) визначає, як взаємодіють два обчислювальні об'єкти, для того, щоб один об'єкт дав можливість другому виконати визначену роботу на користь першого. Ця робота співвідноситься з сервісом, а дії сервісу визначаються мовою описів. Кожна взаємодія самостійна і практично не залежить від іншого. Бізнес-додатки створюються для автоматизації різних бизнес - процесів, але часто без

здійснення в них можливості адаптуватися до потреб, що змінюються, доопрацювання бізнес- процесів в даному середовищі достатньо трудомістке завдання. Все тому, що бізнес- додатки традиційно створюються як одиничні, монолітні, такі, які включають всі інструменти. Тому будь - які зміни в них достатньо дорогі і займають багато часу. У середовищі SOA, додатки створюються у вигляді набору сервісів, кожен з яких має свої завдання і властивості. У міру зміни потреб, деякі сервіси можуть додаватися, деякі видалятися або доопрацьовуватися.

Web - сервіси володіють наступними характеристиками:

- Це Інтернет- додаток, що виконує спеціальні завдання і що підкоряється стандартним специфікаціям.
- Сервіс описується на XML і доступ до нього може бути здійснений за допомогою XML - повідомлень.
- Він може бути анонсований, виявлений і викликаний в розподіленому обчислювальному середовищі.
- Він не залежить від платформи або мови.

Web - сервіс є системою ПЗ, URL якого ідентифікується, а інтерфейси і зв'язки визначені і описані за допомогою XML. Він може бути виявлений іншими системами ПЗ. Ці системи, у свою чергу, можуть взаємодіяти з Web - сервісом, використовуючи XML- повідомлення, що передаються за допомогою Інтернет - протоколів. Мова описів Web - сервісів (WSDL) фактично є заснованим на XML стандартом для опису Web - сервісів. Простий протокол доступу до об'єктів (SOAP) є заснованим на XML стандартним мережевим протоколом для обміну повідомленнями між Web - сервісами (описи W3C).

Забезпечення безпеки в Грід

Компонента Globus Security Infrastructure (GSI) забезпечує захист, включаючий шифрування даних, аутентифікацію (перевірка, що користувач, чи ресурс дійсно являється тим, за кого себе видає) і авторизацію (процедура перевірки, що аутентивікований користувач, чи ресурс дійсно має вимагаючі права доступу) з використанням цифрових сертифікатів X.509.

Можна впевнено стверджувати, що при використанні грід-систем інфраструктура безпеки являється найбільш важливим елементом, так як від неї напряму залежить безпека всієї системи.

Властивості системи безпеки.

Ними являються: конфіденційність, цілісність і аутентифікація. В ідеалі захищений обмін включає всі три складові.

Конфіденційність. Безпечний діалог повинен бути конфіденційним: тільки відправник і отримувач повинні розуміти зміст обміну. Якщо хтось веде прослуховування, він не повинен в діалозі виявити будь-який смисл. Це досягається алгоритмом шифрування-дешифрування.

Цілісність. Безпечна комунікація повинна гарантувати цілісність передаваємих повідомлень. Це означає, що на приймальному кінці мають бути переконані, що прийняте повідомлення в дійсності є те, яке було направлене.

Слід враховувати, що зловмисники можуть перехоплювати повідомлення для модифікації його змісту, а не тільки для використання і зробити це можна навіть не розуміючи його змісту.

Аутентифікація. Безпечна комунікація повинна гарантувати, що сторини обміну є тими, за кого себе видають.

Авторизація. Авторизація відноситься до механізмів, які вирішують чи дозволено користувачу виконувати певну задачу. Авторизація зв'язана з аутентифікацією, тому що потрібно пересвідчитися, що користувач є тим, за кого він себе видає перш ніж вирішувати допускати його до роботи.

Криптографія з відкритим ключем. Забезпечення безпеки в Грід є однією з головних задач і в документації по Грід питанням безпеки відводиться близько 30%. Системи з одним ключем в Грід недопустимі із-за необхідності передачі отримуючі стороні ключа, який тим самим стає доступним третім особам. В грід використовується системи шифрування з відкритим ключем, котрий не потрібно передавати по каналу зв'язку.

Цифровий підпис. Цілісність в системах з відкитим ключем гарантується цифровим підписом. Цифровим підписом є блок даних, який поєднується з повідомленням для вияснення невтручання в повідомлення під час передавання. Цифровий підпис для повідомлення генерується в два кроки:

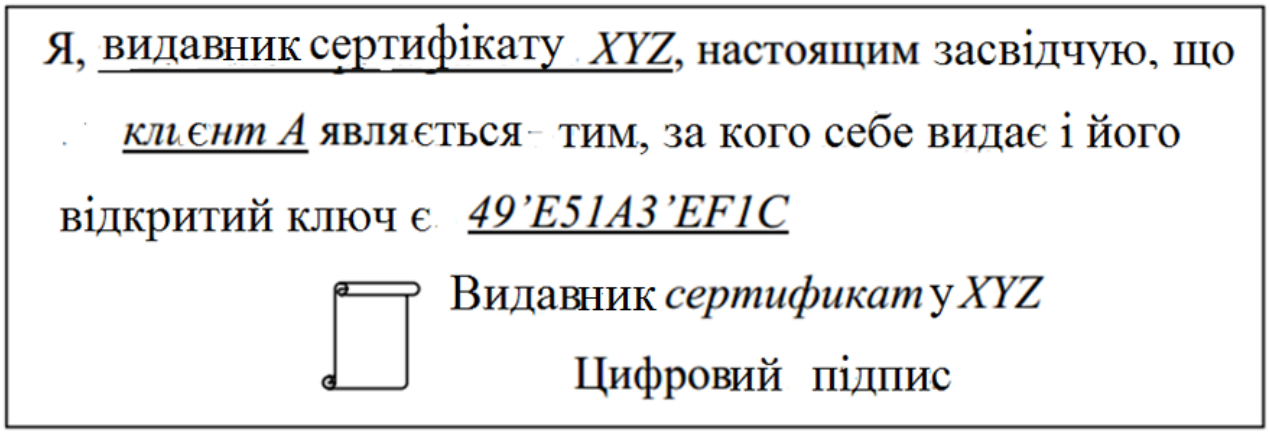
- спочатку генерується цифрове резюме повідомлення: це певна сума повідомлення. Вона завжди менша за повідомлення і навіть найменші зміни в повідомленні дають інше резюме.
- резюме шифрується закритим ключем відправника. Результуюче зашифроване резюме повідомлення називається цифровим підписом.

Цифровий підпис з'єднується з повідомленням і відправляється адресату. Адресат після цього робить наступне:

- використовуючи відкритий ключ відправника, дешифрує цифровий підпис.
- використовує той же самий алгоритм шифрування цифрового резюме, що і відправник, щоб отримати цифрове резюме прийнятого повідомлення.
- порівнює обидва резюме. Якщо між ними є хоч найменша розбіжність, значить було втручання третьої сторони. Таким чином гарантується і цілісність.

Аутентифікація в системах з відкритим ключем

Наведений вище приклад в деякій мірі гарантує аутентичність відправника, оскільки тільки відкритий ключ відправника може дешифрувати цифровий підпис. Але реально відправник може бути не тим, за котрого він себе видає. В багатьох випадках «слабкої аутентифікації» достатньо. Але часом потрібна аутентифікація, при котрій нема сумнівів в відправнику. Це досягається на основі цифрових сертифікатів. Цифровий сертифікат є цифровий документ, який засвідчує, що певний відкритий ключ належить володарю закритого ключа. Цей документ підписується третьою стороною , яка називається certificate authority (CA). Формат CA представлений нижче.



Формат цифрового сертифікату

Звичайно, сертифікат кодується в цифровому форматі. Підпис видавця є в дійсності цифровим підписом, згенерованим за допомогою закритого ключа CA, тому ми можемо провірити цілосність сертифікату, використовуючи відкритий ключ CA. Наявність сертифікату позволяє виконати аутентифікацію. Якщо ви підписуєте ваше повідомлення вашим закритим ключем і посилаєте адресату копію вашого сертифікату, то він може бути впевненим, що повідомленя було послано вами.

X.509 формат. Сертифікат є текстовий файл, який включає багато інформації, представленої в специфічному синтаксисі. X.509 має наступні чотири найбільш важливі речі:

• Subject: це «ім'я» користувача. Воно кодується як унікальне ім'я.
• Subject's public key: Це включає не тільки сам ключ, але і певну інформацію, наприклад, який алгортм використаний для генерації відкритого ключа.
• Issuer's Subject: уникальне ім'я CA.
• Digital Signature: Сертифікат включає цифровий підпис всієї інформації в сертифікаті. Цифровий підпис генерується з використанням відкритого ключа CA.

CA ієрархія. Системи з відкритим ключем будуть мати список всіх CA, котрим можна довіряти, включаючи і ваш CA. Ви повинні вирішити, хто зробить такий список.

Хто підписує CA сертифікати? Відповідь – інший CA. Це позволяє створити ієрархію CA, в котрій ви можете довіряти самому верхньому CA. Ви можете по ступеням ієрархії провірити всі CA, крім останнього, котрому вам приходиться довіряти абсолютно. Сертифікат - відкритий документ, який доступний іншим користувачам для перевірки вашої ідентичності. Об'єднання відкритого і закритого ключів називається в загальному випадку мандатом (credentials) користувача.

Система входу користувача в систему досить складна. Це визначено багатьма факторами, але головною проблемою являється вирішення питання безпеки (загрози вторгнень і атак зловмисників). Аутентифікація і авторизація користувачів являються шляхом для рішення цієї проблеми. Аутентифікаційні

рішення для середовищ віртуальних організацій повинні мати наступні властивості:

• Єдиний вхід. Користувач повинен реєструватися і аутентифікуватися тільки один раз на початку сеансу роботи, отримуючи доступ до всіх доступних ресурсів базового рівня архітектур Грід.

• Делегування прав. Користувач повинен мати можливість запуску програм від свого імені. Таким чином, програми отримують доступ до всіх ресурсів, на котрих авторизований користувач. Користувацькі програми можуть, при необхідності, делегувати частину своїх прав іншим програмам.

• Довірливе відношення до користувача. Якщо користувач запросив одночасну роботу з ресурсами декількох постачальників, то при конфігурації захищеного середовища користувача система безпеки не повинна вимагати взаємодії постачальників ресурсів одного з одним. Для входу в Грід-систему користувач повинен:

• бути легальним користувачем обчислювальних ресурсів в своїй організації;

• мати персональний цифровий сертифікат, підписаний центром сертифікації;

• бути зареєстрованим хоч би в одній віртуальній організації.

Інфраструктуру безпеки в дії можна продемонструвати наступним рис.6.3, де питання безпеки поставлено таким чином: “Створити процеси на вузлах А і В, які потім обмінюються файлами з вузлом С.”

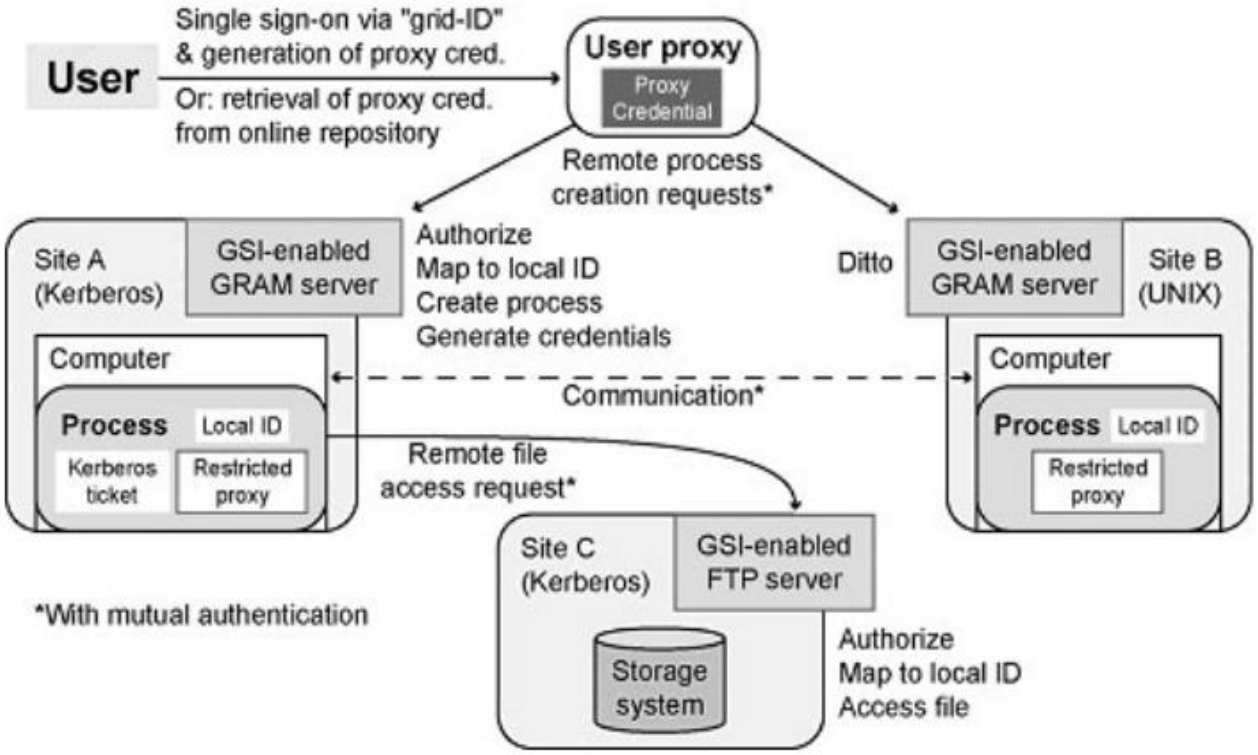


Рис.6. 3 Інфраструктура безпеки

Питання для самоконтролю

1. Що таке Грід?
2. На що націлені Грід - обчислення ?

3. Охарактерізувати програмне забезпечення Грід та його особливості
4. Архітектура протоколів Грід
5. Що таке Fabric Layer?
6. Connectivity Layer – призначення та протоколи
7. Що таке **Ресурсний рівень?**
8. Що таке **Прикладний рівень?**
9. Назвати основні класи протоколів ресурсного рівня
10. Що таке Collective Layer?
11. Назвіть особливості програмування Грід.
12. Призначення системи безпеки Грід.
13. Що таке Globus Security Infrastructure (GSI)?
14. Назвати основні властивості системи безпеки Грід.
15. Що таке конфіденційність, цілісність і аутентифікація?
16. Що таке авторизація?
17. Що таке цифровий підпис?
18. Охарактеризуйте аутентифікацію в системах з відкритим ключем.
19. Що таке Цифровий сертифікат?
20. Що таке X.509 формат?
21 Хто підписує CA сертифікати?
22 Що таке Єдиний вхід і делегування прав?

Література

1. Петренко А.І., Вступ до GRID технологій в науці та освіті: навчальний посібник. - К.: НТУУ «КПІ», 2008. − 120 с.
2. Петренко А.І., Застосування GRID технологій в науці та освіті: додатковий матеріал до вивч. курсу для студ. спец. «Інформаційні технології проектування». − К.: НТУУ «КПІ», 2008. − 144 С.
Режим доступу: http://moodle.ntu-kpi.kiev.ua
3. Петренко А.І., Булах Б.В., Хондар В.С. *Семантичний Грід для науки і освіти*. //– К. : НТУУ «КПІ», 2010. – 180 с.

Література допоміжна

4. Foster I., Kesselman C. The Grid. Blueprint for a new computing infrastructure. San Francisco: Morgan Kaufman, 1999. 677 p.
5.Foster, C. Kesselman, S. Tuecke, J.M. Nick. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. - Morgan Kaufmann Publishers, 2002.
6.Foster, H. Kishimoto, A. Savva, D. Berry et al. The Open Grid Services Architecture. - Global Grid Forum, 2005
7. Introduction to GRID Computing, December 2005. – IBM Redbook, − 241 с. [Електронний ресурс] Режим доступу: www.ibm.com/redbooks
8. Computational problems - Gridcafe. E-sciencecity.org. Retrieved 2013-09-18.

6.2. ***Організація обчислень в Grid- системах***

Суперкомп'ютери, грід-системи та хмарні середовища

Наразі суперкомп'ютери, грід-системи та хмарні середовища є найпотужнішими обчислювальними засобами. Але користувачі часто не усвідомлюють, якого типу застосунки слушно виконувати на одному з цих засобів. Тому далі розглянемо особливості та відміни цих архітектур з точки зору найбільшої придатності для реалізації застосувань.

6.2.1. Порівняння Грід-систем та комп'ютерних кластерів

Під Грід-системою розуміють обчислювальну систему з декількох обчислювальних елементів (СЕ), зв'язаних мережею, які виконують паралельно складне завдання. Така обчислювальна система має три характерні ознаки: її СЕ слабо позв'язані, можуть бути розташовані географічно на великій відстані та можуть бути різного типу (гетерогенними). На відміну від Грід-системи, система з тісно зв'язаних СЕ, така як комп'ютерний кластер (КК), як правило, має пам'ять, що розділяється між СЕ та інші загальні ресурси, які зв'язані між собою через швидкісну систему коротких міжз'єднань.

Як правило, слабко зв'язана система будується з автономних СЕ за принципом майстер – виконавці. Причому СЕ-майстер розділяє велике завдання на незалежні підзавдання, розподіляє їх серед СЕ-виконавців, запускає їх на виконання, збирає від них результати та обробляє їх для формування кінцевого результату.

Грід-система – це система з слабко позв'язаних СЕ. Ці СЕ можуть виконувати окремі незалежні завдання, допоки операційна система (ОС) не об'єднає їх для виконання функції Грід-системи. Причому такий СЕ може продовжувати виконання свого автономного завдання у багатопотоковому режимі, виконуючи функції частини Грід-системи. Цей слабкий зв'язок між СЕ дає змогу географічно віддаленного розміщення СЕ. Оскільки СЕ зв'язані між собою через стандартну мережу (найчастіше – Інтернет), дійсно, не має значення відстань між цими СЕ. Така географічно розподілена мережа має ряд переваг. Загальне завдання у такій мережі вирішується менш витратно, оскільки вона дає змогу краще завантажити недовантажжені СЕ. Ця мережа не має проблем з електричним живленням, оскільки різні СЕ живляться від різних електричних мереж, які не здатні відмовити одночасно. СЕ мережі можуть мати різні ОС та різні продуктивності, хотя багато з Грід-систем побудовані на однотипних СЕ.

Високопродуктивні суперкомп'ютери аналогічні Грід-системам в тому, що серед них багато зв'язаних систем, як наприклад, КК. Але ця зв'язаність є щільною. Якщо між СЕ у Грід-системі передача даних відбувається як асинхронна пересилка повідомлень з підтвердженням їх приймання, то у суперкомп'ютерах ця пересилка відбувається синхронно без очікування такого підтвердження. Через це СЕ суперкомп'ютерів мають бути зконцентровані в одному місці і з'єднані між собою швидкодіючими каналами зв'язку, а не через звичайну локальну мережу чи Інтернет. Причому ці СЕ мають бути однорідними.

6.2.2 Програмування Грід-систем та комп’ютерних кластерів

Відміни між архітектурами КК та Грід-систем роблять суттєво різними технології створення програмного забезпечення для них. Тому досі створення методів розробки матзабезпечення для цих архітектур залишається в галузі досліджень та розробки.

Однією з ключових концепцій в написанні застосувань для цих архітектур є ідея батьківських і дочірніх процесів, яка полягає у наступному. Велике завдання аналізується на можливість розділення його на дрібніші завдання. Диспетчер розподіленого обчислювального середовища розділяє завдання на частини, які розподіляються серед вільних СЕ. Перш ніж продовжити обробку, головна задача ("батько"), чекає дані, які повинні бути повернуті від кожного з дрібних підзадач («дітей»). Найбільш важливою проблемою при програмуванні Грід-систем чи КК є кодування у програмі способу, який розділяє велике завдання на дрібні під завдання так , щоб дочірні процеси завершувались якомога швидше. Таке програмування є досить складним. Найбільшою проблемою є нова архітектура, для якої програміст повинен думати в термінах батьківських та дочірніх процесів, які розподіляються по СЕ при певних умовах. Тут програміст має турбуватись не про зацикленість програми, а про баланс кількості та складності дочірніх процесів та кількості вільних ресурсів. Може здаватися, що Грід-система та КК мають однакові архітектури з точки зору програмування, але Грід-система є більш складною для програмування через асинхронність пересилок даних, які мають різні затримки. Крім того, слід мати на увазі, що як канали зв’язку, так і деякі СЕ Грід-системи можуть вимикатись на тривалий час. Тому користувацька програма повинна мати властивість динамічно змінювати хід свого виконання.

6.2.3 Планування завдань в Грід-системах та комп’ютерних кластерах

Як Грід-система, так і КК мають у своїх ОС планувальники завдань. Планувальник повинен мати не тільки інформацію про програми, які запускаються, але й про алгоритми, які керують породженням дочірніх процесів, про наявність вільних ресурсів в кожній ділянці обчислювального середовища, кількість яких може рахуватися сотнями та тисячами. Крім того він займається перевіркою аутентичності завдань, розподілом їх по ресурсах і фактичним ходом їх виконання.

В області високопродуктивних обчислень часто використовується планувальник, створений у Job Submission Description Language Working Group (JSDL-WG) та Maui Scheduler. Причому перший набув поширення серед Грід-систем, а другий – серед КК.

JSDL-WG – це мова управління завданнями, яка орієнтована на Грід-системи та є незалежною від будь-якої мови програмування, яка може бути використана користувачем. У описі JSDL можна використовувати код XML, щоб визначити різні аспекти запиту ресурсів, що робить його відкритим стандартом. Це поняття відкритого доступу є критичним при розгляданні переходу від програмування Грід-систем до програмування хмарних обчислень.

Планувальник Maui Scheduler був розроблений в 1990-х роках як планувальник з відкритим вихідним кодом для ОС високопродуктивних КК, які використовують ОС Unix. Maui Scheduler дозволяє адміністраторам встановлювати розклад і пріоритети, резервувати ресурси для конкретних програм, або їх частин, а також виконувати інші завдання, які роблять системи більш функціональними.

Після поширення Maui Scheduler була розроблена в Центрі високопродуктивних обчислень Університету Юта його комерційна версія Moab. Вона додатково підтримує різні ОС, у тім рахунку Linux, Mac OS X та Windows. Також вона має такі функції, як менеджер робочих навантажень, менеджер кластеру, порталу доступу і керування ресурсами, які не доступні в Maui Scheduler. Ця тенденція, коли за основу береться програмне забезпечення з відкритим кодом і дороблюються програми, які додають можливостей, можна побачити у розвитку хмарних обчислень. У планувальнику Moab Scheduler завдання розглядаються з урахуванням політики справедливості пріоритетів, яка враховує тривалість знаходження в черзі, можливостей розширення завдання, історію використання СЕ користувачем і яка конфігурується адміністратором КК. Планувальник Moab Scheduler дає дозвіл, щоб певні СЕ були зарезервовані для певних користувачів, груп, рахунків, або проектів, що мінімізує кількість витрачених ресурсів комп'ютера, з урахуванням очікуваного простою.

Зараз Moab Scheduler використовується як інструмент для дослідження обчислювальних процесів в КК. Цей планувальник збирає велику кількість статистичних даних і може проаналізувати цю інформацію, щоб визначити ефективність того чи іншого підходу до планування, використання наявних ресурсів[1].

6.2 .4 Особливості планування у Грід-системах

Перед тим, як розглядати обчислювальні процеси у хмарному середовищі, слід ознайомитись з процесом планування завдань в обчислювальній мережі Грід-систем. Слід зазначити, що велику роботу по стандартизації планування Грід-систем зроблено на форумі Open Grid Services Architecture (OGSA)[2]. Також значний внесок зробила фірма IBM під час розробки КК *Deep Blue*, яка випустила документацію в серії Red Book.

Процес планування в Грід-системі можна розділити на три етапи, які мають 11 кроків і описуються нижче.

Етап I: Визначення ресурсів

Крок 1: Фільтрація авторизуючих запитів.

Перед тим, як розмістити своє завдання у Грід-системі, користувач повинен пройти аутентифікацію. Крім того, інформація, необхідна для аутентифікації, включає в себе профіль користувача, що описує ресурси, які можуть бути використані, інформацію про оплату використання ресурсів. Цей профіль може також включати в себе деякі облікові дані для одного, або декількох ресурсів, які підключаються до мережі. Код одноразового доступу, як правило, перевіряється на початковому етапі фільтрації авторизуючих запитів, дозволяючи обліковим даним, які зберігаються у одному СЕ в якості

аутентифікаційних документів, бути застосованими для інших СЕ в рамках кластеру.

Крок 2: Визначення застосування.

На цьому етапі користувач вказує мінімальний набір ресурсів, які розглядає планувальник. Ця інформація включає в себе об'єм необхідної пам'яті, робочого простору (місця на дисках), перелік спеціалізованих бібліотек тощо і описується як скрипт для планувальника. Вона потрібна для того, щоб планувальник знав, які ресурси необхідні для завдання, щоб воно було виконано успішно.

Поширення стандартного опису XML, який використовується для обміну даними, зробило опис застосувань більш стандартизованим та відкритим для більшості кластерів і мереж. Такий опис не потрібно виконувати вручну, бо він, як правило, формується автоматично за допомогою відповідних додатків до Грід-системи.

Крок 3: Виділення мінімальних вимог до середовища.

В залежності від ресурсів, заявлених авторизованим користувачем, планувальник намагається знайти перший вільний слот (СЕ та інтервал часу), у якому буде виконуватися завдання. Незалежно від завдання і розміру КК, цей крок пов'язаний з вирішенням задачі пошуку відповідності між потребами в ресурсах і наявними ресурсами, щоб задовольнити потреби користувачів у встановлені терміни.

Етап II: Вибір системи СЕ

Крок 4: Збір інформації.

Хоча оптимальне рішення задачі призначення завдання на певний набір апаратних засобів може бути знайдене за одним з відомих алгоритмів, але місцеві правила можуть змусити користувача використати інший набір ресурсів. Наприклад, для університетської Грід факультети університету, здавалось, мають рівноправний доступ до Грід-ресурсів. Але часто окремі факультети, які , наприклад, виконують роботи по удосконаленню системи, можуть мати більше прав доступу і їм планувальник буде виділяти більшу квоту.

Слід зазначити, що існує можливість для зміни ресурсів до того моменту, коли завдання буде представлене для вирішення. Значна кількість досліджень робиться у галузі прогнозування розподілу ресурсів, щоб створити адаптоване виділення ресурсів, що дозволить збільшити точність прогнозу цих ресурсів.

Крок 5: Вибір системи СЕ.

Алгоритми, які вирішують, яку множину СЕ використовувати для конкретного завдання, варіюються по складності. Але у багатьох проектах розробки планувальників використовується одна і та сама тенденція. Це використання методів, основаних на алгоритмах пошуку паропоєднань Condor, обчислювальної економіки та багатьох інших.

Алгоритм пошуку паропоєднань Condor заснований на класичній схемі оголошень. Обчислювальні ресурси, – це за анологією – "продавці", які рекламують свої послуги та можливості через клас ClassAds, де вони можуть також вказати запитувану ціну для ресурсу. Користувачі та завдання – то

"покупці" , які можуть створювати власні класи ClassAds, де зазначаються їх вимоги до ресурсів та заявки оплати, які вони готові платити, щоб їхні вимоги були виконані. Алгоритм Condor перебирає класи ClassAds від продавців і покупців і робить кращі паропоєднання, на основі вказаних специфікацій. Пошук паропоєднання є одноразовим статичним процесом, хоча у будь-який момент за алгоритмом Condor може відбутися повторне сканування заявок та ресурсів, щоб визначити, чи можна знайти кращі паропоєднання.

Алгоритми обчислювальної економіки включають в себе різні методи, які використовуються, щоб вирішити, які процеси призначити на певні ресурси. Ці методи запозичені з теорії класичної економіки, щоб усунути нестабільність вирішення завдання призначення, яка виникає при певних обставинах. Після застосування алгоритму обчислювальної економіки, ресурси і завдання можуть бути динамічно переоцінені в залежності від попиту на ресурси і попиту на результати обчислень, а центральний планувальник постійно приймає рішення щодо запуску завдань, які ґрунтуються на останніх оцінках. Система планування на основі алгоритмів економіки повинна мати на увазі що в будь-який момент ресурси мають використовуватися найбільш ефективно, а саме завдання повинне виконуватись з найбільшою ефективністю.

Етап III: Виконання завдання

Крок 6: Попереднє резервування ресурсів

Попереднє резервування ресурсів у Грід-системі може бути таке саме за складністю, як організація об'єднання комп'ютерів через локальну мережу, або як вирішення задачі масового планування. Але більшість користувачів замовляє ресурси у Грід-системі з надлишком, тобто «про всяк випадок» і це також слід враховувати.

Крок 7: Представлення завдання

Часто завдання користувача представлене з розбіжностями відносно стандартів, на які налаштована дана Грід-система. Тому представлення такого завдання потребує узгодження. Через це Грід-системи, основані на відкритому матзабезпеченні, завжди будуть позаду комерційних Грід-систем через недостатню увагу до стандартизації оформлення завдань.

Крок 8: Підготовка завдання

Як тільки починається виконання завдання, ресурси, такі як набори даних будуть переміщені в тимчасові файли і батьківський процес починає заповнювати інформацію для кожного дочірнього процесу. При цьому для розсилання даних використовуються FTP (File Transfer Protocol), SCP (Secure Copy Protocol), і SFTP (Secure File Transfer Protocol), а також протокол Torrent.

Крок 9: Моніторинг виконання завдання

Може так статись, що якийсь дочірній процес "зависне", наприклад, через виключний стан, відсутність вхідних даних з відповідним простоюванням ресурсів. У гіршому випадку така поведінка може тягнути за собою "зависання" усього завдання. Тому планувальник повинен відслідковувати подібні ситуації і переривати виконання завдання.

Крок 10: Завершення роботи

Бідь-яке завдання має метою розрахувати певного роду результати і припинити своє виконання, що має бути зафіксовано планувальником.

Крок 11: Очищення середовища

Планувальник турбується про стирання тимчасових файлів у разі, коли завдання користувача не зробила це самостійно.

Програмні рішення Грід

В Грід використовуються основні елементи технології WebServices. Це позволяє оформити функції Грід-технлогії (службові і ресурсні) в вигляді веб-сервісів, використовуючи для роботи з ними всі стандартні технології WebServices. На рис.6.4 схематично показаний простий сервісно-орієнтований грід, в якому сервіси використовуються і для віртуалізації і для забезпечення інших функціональних можливостей Грід.

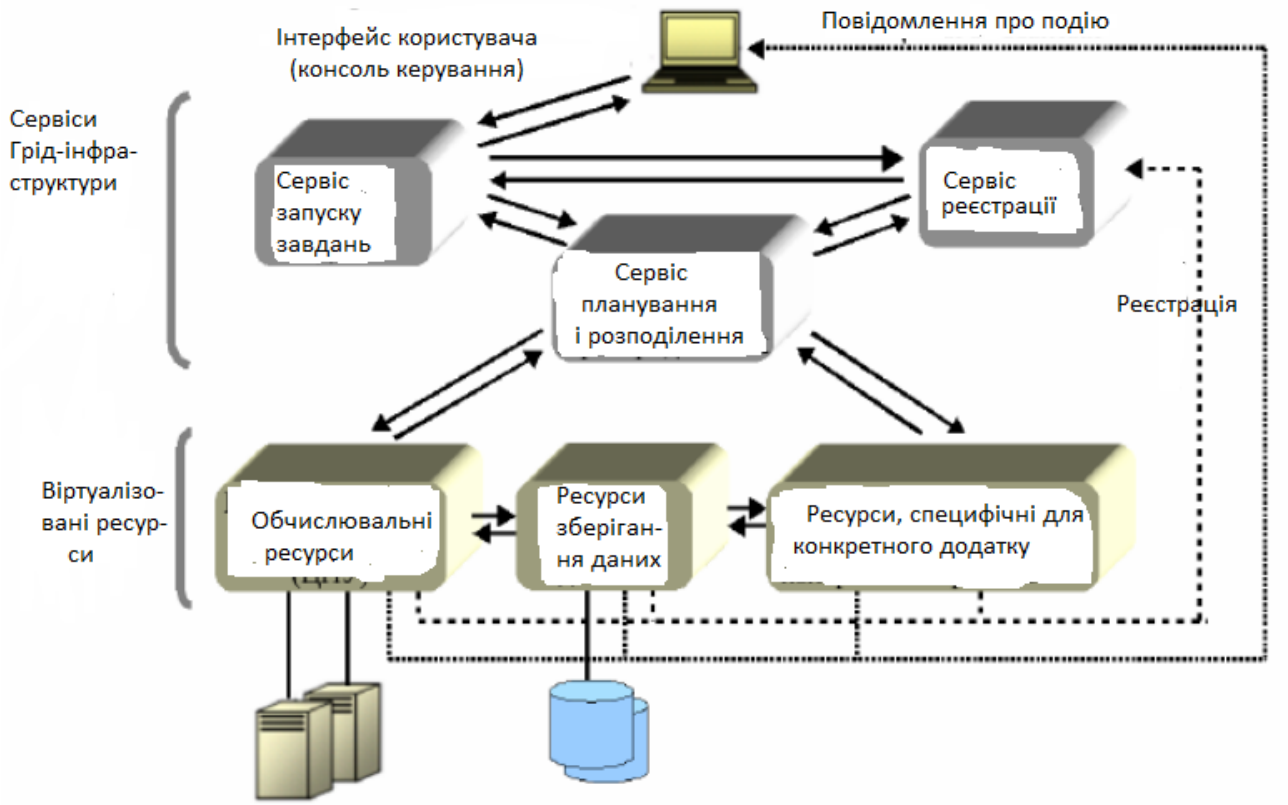


Рис.6.4 Схема сервісно-орієнтованого грід

На схемі показана єдина консоль і для запуску завдань в Грід і для керування Грід-ресурсами. Програмне забезпечення інтерфейсу користувача (консолі) звертається до сервісу реєстрації для отримання інформації про існуючі Грід-ресурси. Після цього користувач за допомогою консолі входить в контакт з сервісами, які представляють (віртуалізують) кожний ресурс, щоб запросити періодичне отримання даних про роботу ресурсів і отримати повідомлення про суттєвих змінах в їх стані (наприклад, якщо ресурс стає недоступним, чи сильно завантаженим).

Користувач направляє запит на запуск в службу запуску, яка передає запит службі розподілення завдань (яку ще називають планувальником). Ця служба контактує зі службою, яка представляє додаток і запрошує інформацію про вимоги до ресурсів виконання завдання. Далі служба розподілення запитує (направляє запит) в служби реєстрації інформації про всі необхідні ресурси в Грід і напряму контактує з ними, щоб пересвідчитись в їх доступності.

Якщо необхідні ресурси доступні, планувальник вибирає найкращу доступну сукупність ресурсів і передає інформацію про них сервісу додатку з запитом на початок виконання. В іншому випадку планувальник ставить завдання користувача в чергу і виконує його, коли необхідні ресурси стають доступними. Коли виконання завдання закінчується, сервіс додатку повідомляє про результат планувальнику, який повідомляє про це сервіс запуску завдань. Сервіс запуску завдань, в свою чергу, увідомлює користувача. Слід зауважити, що наведений приклад дуже спрощений: функціонування реального Грід набагато складніше.

В середовищі Грід явним чином присутні наступні елементи:

- Програми користувача
- Ресурси (“залізо”, ОС, кластерне ПЗ та ін.)
- Проміжне програмне забезпечення (Middleware), яке виступає в ролі посередника між користувацькими програмами і ресурсами.

Middleware включає великий обсяг програмного забезпечення, яке створюється великими організаціями і строго стандартизується, щоб забезпечити його використання різними користувачами. В число найбільш відомих пакетів middleware входять:

•Globus Toolkit. Він вважається американським напрямком;

•gLITE. Являється європейським проектом і підтримується Європейським центром ядерних досліджень (CERN).

•UNICORE. Теж являється європейським проектом.

Вони розрізняються між собою, але мають багато спільного, оскільки використовують, як правило, систему Globus Toolkit.

Різні middleware–системи можуть розрізнятися по рівню. Наприклад, gLITE ставиться поверхи Globus Toolkit.

В глобальних Грід-системах в якості middleware використовують інструментарій Globus Toolkit, розроблений американськими вченими, який став de facto світовим стандартом. Він включає в себе спеціальний протокол на основі HTTP для використання обчислювальних ресурсів GRAM (Grid Resource Allocation Management); розширену версію протоколу для передачі файлів GridFTP; службу безпеки GSI (Grid Security Infrastructure); розподілений доступ до інформації на основі протоколу LDAP; віддалений доступ до даних через інтерфейс GASS (Globus Access to Secondary Storage). Ці служби дозволяють побудувати повнофункціональну Грід- систему.

Далі коротко розглянемо розроблене і використовуємe middleware Грід.

Програмне вирішення GLOBUS

Набір програм Globus Toolkit (GT) – програмний продукт з відкритим початковим кодом і набором бібліотек, розроблений в США в національній лабораторії. Він містить набір стандартних блоків і інструментів, які можуть бути використані розробниками і системними інтеграторами. За декілька років вийшли кілька версій програми Globus: оригінальна – в кінці дев'яностих, GT2 – в 2000, GT3 – в 2003, і GT4 – в 2005, GT5, GT6. Версія GT2 послужила базисом для безлічі Грід-розробників по всьому світу. GT3 – стала першою повноцінною реалізацією інфраструктури Грід, побудованої на технології Web -

сервісів, з використанням проміжної ланки GGF's OGSI. GT4 – перша версія, повністю сумісна з основними Web - сервісами так само, як GRID - сервіси засновані на WSDL і WSRF. Інструментарій підтримує операційні системи Linux (RHEL 5-7, CentOS 5-7, Debian 6-7 та ін.), MacOS (10.6 та новіші) та Windows (Windows 7 та новіші).

В стабільному релізі GT 4 Final зафіксовані всі зовнішні інтерфейси.

На сайті Globus Alliance приводиться опис складу служб, відомості про них, плани і стан реалізації, але підтримка завершилася в січні 2018 року, натомість користувачам пропонується використовувати платформу Globus Connect [4]).

На рис. 6.5 приведені компоненти GT 4.

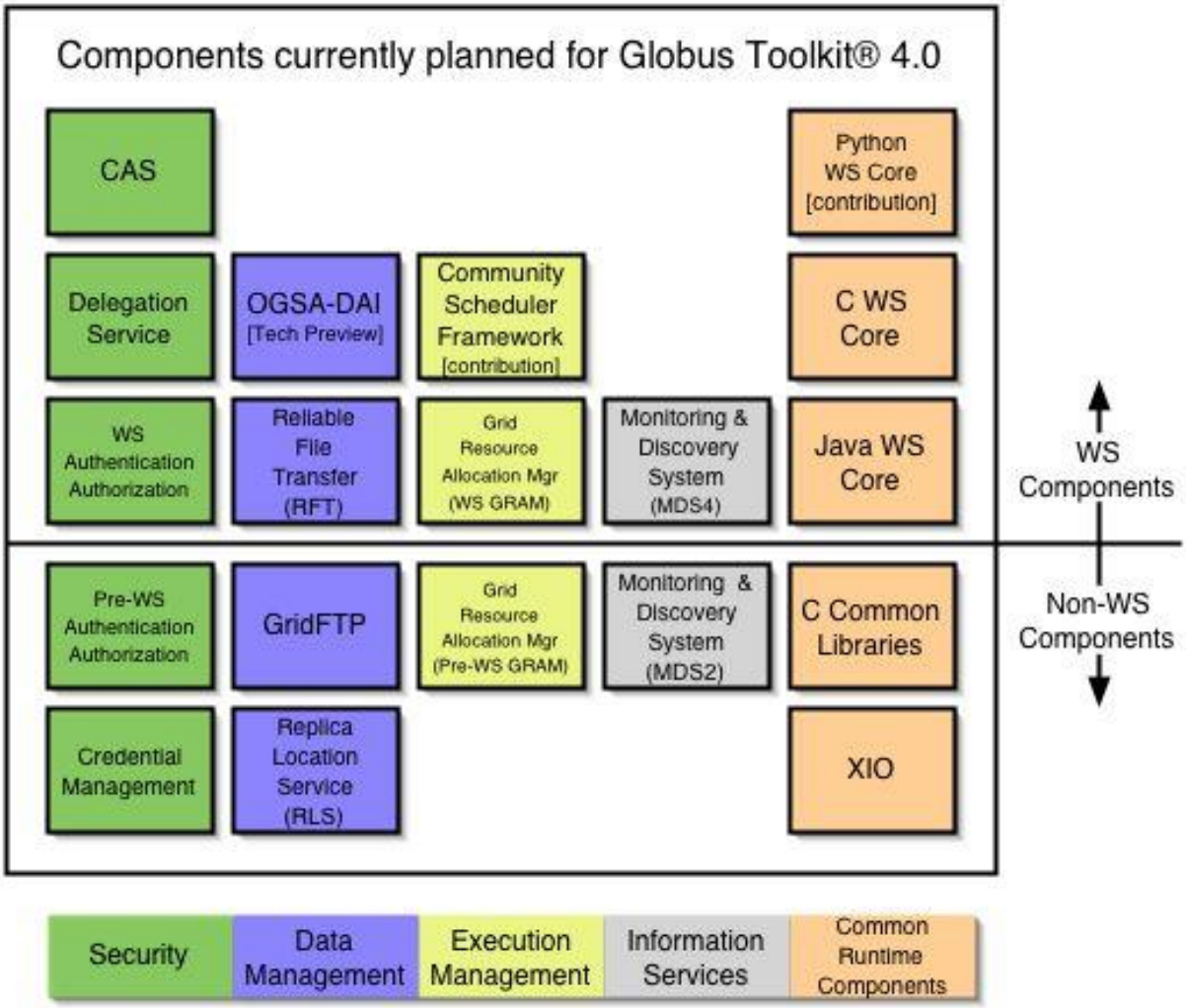


Рис. 6.5. Компоненти Globus Toolkit GT4

Для управління виконанням пакет надає можливість виявлення і управління ресурсами Грід, управління робочим простором і засобом планувальника співтовариства користувачів.

Для забезпечення безпеки пакет надає сервіси аутентифікації і авторизації, надання віддаленого доступу і авторизації співтовариств користувачів.

Для управління даними в програмі закладені функції надійної передачі файлів, інтеграції і доступу до даних і їх тиражування. Для забезпечення інформаційних служб, в програму закладені функції моніторингу і виявлення

різних сервісів системи. Для підтримки спільної роботи система містить різні ядра Web – сервісів, бібліотеки і розширені функції підтримки введення/виводу.

GT4 містить набір стандартних служб:

1. Управління завданнями: Пакет програм виявлення і управління ресурсами (GRAM);.
2. Надійна файлопередача (RFT);.
3. Делегування функцій.
4. Система моніторингу і виявлення вільних ресурсів – індекс (MDS - index);
5. Система моніторингу і виявлення – MDS - trigge.
6. Система моніторингу і виявлення – збір даних (MDS - aggregate);
7. Авторизація співтовариства (CAS);
8. Інтеграція і доступ до даним (OGSA - DAI);
9. Робота в реальному масштабі часу.

Слід зазначити, що Globus Toolkit не містить брокера ресурсів, залишаючи завдання його реалізації розробникам, що створюють системи Грід на його основі.

Склад Globus Toolkit 5.0 (GT5) представлений на рис.6.6.

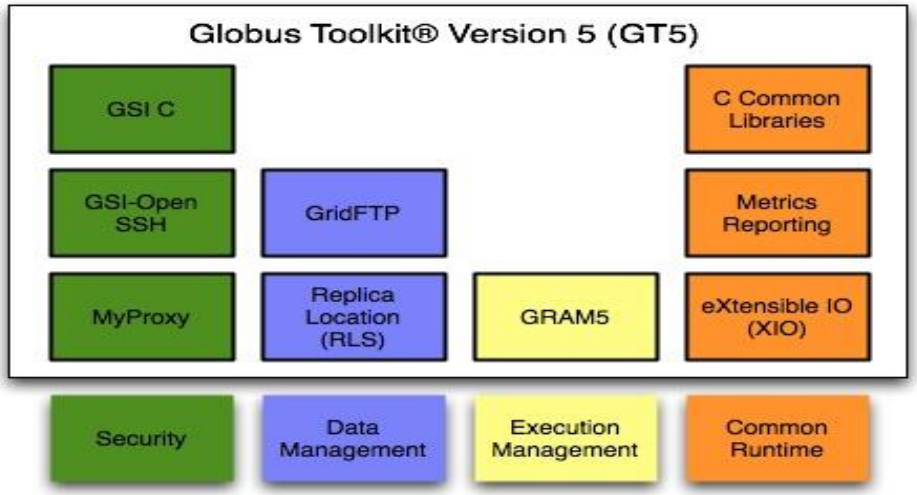


Рис.6.6. Склад програмного забезпечення GlobalToolkit GT5

У порівнянні з версією 4, версія 5.0 зазнала наступних основних змін [3]:

- вдосконалення підтримки управління кореневими сертифікатами у компоненті MyProxy v5.0;
- клієнт GSI-OpenSSH типово намагаються виконувати лише автентифікацію GSI (при цьому підтримка інших методів зберігається);
- в RLS досконалена підтримка 64-бітних операційних систем, сумісність із специфікаціями ODBC при збереженні повної сумісності з GT 4 RLS;
- GRAM5, у порівнянні з GRAM2 та GRAM4, є одночасно надійним та масштабованим, при цьому збережена сумісність з GRAM2;
- покращена система журналювання в GRAM5, знижено споживання ресурсів щонайменше в 10 разів.

Версія GT6 зберігає сумісність з версіями 5.0 та 5.2 на рівні протоколів взаємодії, джерельного коду (API) та двійкових програмних інтерфейсів (ABI). Основні компоненти:
- Grid Security Infrastructure in C (GSI C) – надає інтерфейси прикладного програмування та інструменти забезпечення автентифікації, авторизації та управління сертифікатами;
- MyProxy – програмне забезпечення управління інфраструктурою публічних ключів X.509 (PKI) облікових записів (сертифікати та приватні ключі);
- GridFTP – високоефективний, безпечний та надійний протокол передачі даних, що є розширеним варіантом протоколу FTP;
- Grid Resource Allocation and Management (GRAM5) – компонент, що розподілення, моніторинг та управління завданнями в ресурсах мережі Grid;
- GSI-OpenSSH – модифікована версія OpenSSH, яка надає підтримку автентифікації та делегування з використанням X.509 проксі-сертифікатів;
- C Common Libraries – загальні бібліотеки для мови C;
- XIO – розширювана бібліотека введення/виведення написана мовою C, яка надає єдиний інтерфейс прикладного програмування (open/close/read/write) з підтримкою різноманітних протоколів взаємодії додатків. Підтримка протоколів інкапсульована у вигляді "драйверів", до стандартних входить підтримка: TCP, UDP, файли, HTTP, GSI, GSSAPI_FTP, TELNET та черги;
- SimpleCA – надає спрощену реалізацію центра сертифікації, який може видавати сертифікати X.509 для користувачів та служб Globus Toolkit.

Програмне забезпечення LCG

Для побудови повністю функціональної Грід -системи необхідне програмне забезпечення проміжного рівня, що побудоване на базі існуючих інструментальних засобів і надає високорівневі сервіси завданням і користувачам. Прикладом такого програмного забезпечення може служити ПЗ LCG (LHC Computing Grid), що розроблявся в Європейському центрі ядерних досліджень (CERN). Спочатку метою проекту LCG була розробка повністю функціонуючої Грід -системи на базі Globus Toolkit для обробки даних у фізиці високих енергій. З часом область застосування LCG розширилася, і в даний час це – один з найпоширеніших пакетів ПО Грід.

Пакет LCG складається з декількох частин, званих елементами. Кожен елемент є самостійним набором програм (одні і ті ж програми можуть входити в декілька елементів), які реалізовують деякий сервіс і призначений для встановлення на комп'ютер під управлінням ОС Scientific Linux.

Нижче перераховані основні елементи LCG і їх призначення:

- **CE (Computing Element)** – набір програм, призначений для встановлення на вузол, який керує, обчислювального кластера. Даний елемент надає універсальний інтерфейс до системи керування ресурсами кластера і дозволяє запускати на кластері обчислювальні завдання;
- **SE (Storage Element)** – набір програм, призначений для встановлення на вузол зберігання даних. Даний елемент надає універсальний інтерфейс

до системи зберігання даних і дозволяє керувати даними (файлами) в GRID-системі;

- **WN (Worker Node)** – набір програм, призначений для встановлення на кожен обчислювальний вузол кластера. Даний елемент надає стандартні функції і бібліотеки LCG завданням, що виконуються на даному обчислювальному вузлі;
- **UI (User Interface)** – набір програм, які реалізовують призначений для користувача інтерфейс Грід-системи (інтерфейс командного рядка). У цей елемент входять стандартні команди керування завданнями і даними, деякі з яких розглянуті в наступному розділі;
- **RB (Resource Broker)** – набір програм, які реалізовують систему встановлення завантаженням (брокер ресурсів). Це найбільш складний (і об'ємний) елемент LCG, що надає всі необхідні функції для скоординованого автоматичного керування завданнями в Грід -системі;
- **PX (Proxy)** – набір програм, що реалізовують сервіс автоматичного оновлення сертифікатів (myproxy);
- **LFC (Local File Catalog)** – набір програм, що реалізовують файловий каталог GRID-системи. Файловий каталог необхідний для зберігання інформації про копії (репліках) файлів, а також для пошуку ресурсів, які містять необхідні дані;
- **BDII (Infornation Index)** – набір програм, що реалізовують інформаційний індекс GRID-системи. Інформаційний індекс містить всю інформацію про поточний стан ресурсів, отримувану з інформаційних сервісів, і необхідний для пошуку ресурсів;
- **MON (Monitor)** – набір програм для моніторингу обчислювального кластера. Даний елемент збирає і зберігає в базі даних інформацію про стан і використання ресурсів кластера.
- **VOMS (VO Management Service)** – набір програм, які реалізовують каталог віртуальних організацій. Даний каталог необхідний для керування доступом користувачів до ресурсів Грід-системи на основі членства у віртуальних організаціях.

На один комп'ютер можлива установка відразу декількох елементів LCG, якщо це дозволяють його потужності (об'єм пам'яті і продуктивність). Мінімальна кількість вузлів, необхідних для розгортання повного набору ПЗ LCG, рівна трьом. Слід відмітити, що установка всіх сервісів на один вузол, хоча і можлива технічно, але настійно не рекомендується. Брокер ресурсів, з міркувань безпеки, слід розташувати на окремому вузлі. Обчислювальні вузли також слід виділити окремо, оскільки навантаження, що створюється на них працюючими завданнями, приведе до дефіциту ресурсів для решти сервісів. Решта всіх елементів може бути встановлені спільно.

У основі ПО LCG лежать розробки, виконані в рамках європейського проекту EDG (European DataGrid), які використані в більш функціональній інфраструктурі програмного забезпечення, що носить назву gLite, зі збереженням сумісності.

Важливо відзначити, що все програмне забезпечення в рамках проекту LCG, може вільно використовуватися. На основі цього програмного забезпечення можливе створення національних і регіональних Грід -систем для ефективного розподілу локальних ресурсів. LCG став технологічною базою для інфраструктури в рамках проекту EGEE.

Middleware gLite

Навіть враховуючи появу в Globus Toolkit нових служб (CAS, RLS), слід визнати, що головний напрям його розвитку – засоби розробки і підтримки служб. Основна відмінність комплексу gLite в тому, що крім інструментальних засобів, в нього входить ширший набір служб. gLite істотно спирається на досвід ряду великих європейських проектів: EDG, LCG, Alien, NorduGrid і створювався колективно - в його розробці брали участь більше 80 фахівців з 11 дослідницьких центрів.

Приведемо огляд основних підсистем gLite.

Обчислювальний елемент (Computing Element - CE) - це служба, яка представляє ресурсний вузол GRID і що виконує на нім функції керування завданнями (запуск, видалення і так далі). Звернення до CE можуть надходити або від інтерфейсу користувача, або від Менеджера завантаження (Workload Manager -wm), який розподіляє завдання по безлічі CE. У glite функціональність CE розширена в порівнянні з аналогічною службою Lcg-2. Якщо в Lcg-2 CE може працювати тільки у відповідності до Push моделі (WM самостійно ухвалює рішення про посилання завдання на CE), то в glite можливий режим роботи CE також і в Pull моделі, коли CE запрошує завдання у WM.

Крім функцій управління завданнями, CE також виробляє інформацію про стан ресурсів. У Push моделі її публікує інформаційна служба, і вона використовується WM для вибору CE, на якому запускатиметься завдання. У Pull моделі інформація вбудовується в посилане WM повідомлення "CE доступний".

Підсистема управління даними (Data Management Subsystem - DM) включає три служби, що підтримують доступ до файлів: елемент пам'яті (Storage Element - SE), служби каталога (Catalog Services - CS) і диспетчер даних (Data Scheduling -ds). Всі служби працюють з даними на файловому рівні, в протилежність, наприклад, системам баз даних, які оперують такими елементами як записи і поля. У розподіленому середовищі Грід призначені для користувача файли можуть зберігатися в безлічі екземплярів - реплік, розміщених в різних місцях, і завдання CS і DS полягає в тому, щоб зробити процес управління репліками прозорим для користувача, так щоб додатки діставали доступ до файлів по їх іменах, або дескрипторах метаданих.

Доступ до даних файлів реально відбувається через SE, але DM підтримує також концепцію віртуальних наборів даних. Це відкриває нові цікаві можливості, засновані на абстракції глобальної файлової системи: при навігації по файлах клієнтське застосування може бути влаштоване як командна оболонка Unix, використовуючи команди зміни директорій, проглядання файлів і тому подібне Захист файлів забезпечується в DM списками контролю доступу ACL (Access Control Lists).

Підсистема обліку (Accounting Subsystem - DGAS) акумулює інформацію про використання ресурсів Грід окремими користувачами, групами користувачів і віртуальними організаціями. Зібрана інформація дозволяє побудувати загальну картину діяльності в Грід, на основі якої може формуватися політика розподілу ресурсів і стягуватися плата за їх використання.

Підсистема протоколювання (Logging and Bookkeeping - LB) відстежує кроки обробки завдання, виконуються в різних точках Грід, фіксуючи події (запуск, розподіл на відповідний CE, початок виконання і так далі), що відбуваються і запам'ятовуючи їх. Інформація про події (протокол) поставляється компонентами WM і CE, для чого в ці компоненти вбудовуються звернення до LB. Протоколи збираються в два прийоми. Спочатку події передаються в локальну службу (Locallogger) і записуються у файл на диску. Locallogger відповідає за передачу протоколу одному з серверів зберігання (Bookkeeper), який "збільшує" події, даючи загальну картину зміни станів завдання (Submitted, Running, Done.). Крім того, Bookkeeper зберігає різні атрибути завдання: його опис (JDL); CE, на якому воно виконувалося; коди завершення і так далі. Як протокол станів, так і протокол подій можна отримати або за допомогою спеціального інтерфейсу WM, або через повідомлення при певних змінах стану, наприклад, при закінченні завдання.

Підсистема інформаційного обслуговування і моніторингу Грід (Relational Grid Monitoring Architecture - R-GMA) вирішує задачу збору і управління даними про стан Грід, отримуючи інформацію від безлічі розподілених джерел - постачальників. У її основі лежить розроблена однією з груп Global Grid Forum (GGF-PERF) схема "Споживач-Постачальник", яка описує спосіб взаємодії цих компонентів. Оскільки схема достатньо загальна, вона застосовна як для зберігання даних про Грід (які ресурси і служби доступні, які їх характеристики), так і для моніторингу додатків. R-GMA є реляційною реалізацією цього загального підходу. За наявності безлічі розподілених постачальників з погляду інформаційних запитів R-GMA діє як одна велика реляційна база даних. "Реляційність" виявляється у формі представлення даних: постачальники оголошують про склад публікованої інформації за допомогою конструкції SQL CREATE TABLE, публікують її за допомогою SQL INSERT, а споживачі отримують дані через SQL SELECT.

Підсистема управління завантаженням (Workload Management System - WMS) складається з ряду компонентів, відповідальних за розподіл завдань між ресурсами Грід, а також що забезпечують управління завданнями. Центральною компонентой є Менеджер завантаження (WM), який отримує від своїх клієнтів запити по управлінню завданнями. Зокрема, обробляючи запит типу "запуск" WM визначає відповідний для виконання CE, зважаючи на вимоги і переваги, задані в описі завдання.

Система безпеки. Вона розглядається як засіб захисту Web-служб і реалізована у вигляді додаткових модулів, що розміщуються в контейнерах (Apache Axis, Tomcat). Розроблені пропозиції по архітектурі безпеки: https://edms.cern.ch/document/487004/ і сформульовані основні цілі: модульність,

розширюваність, відповідність стандартам Web-служб, що розвиваються (Ws-security).

Інфраструктуру безопеки в дії можна продемонструвати наступним малюнко, де питання безпеки поставлено таким чином: “Створити процеси на взлах A і B, які потім обмінюються файлами з вузлом C.” (рис.6.7)

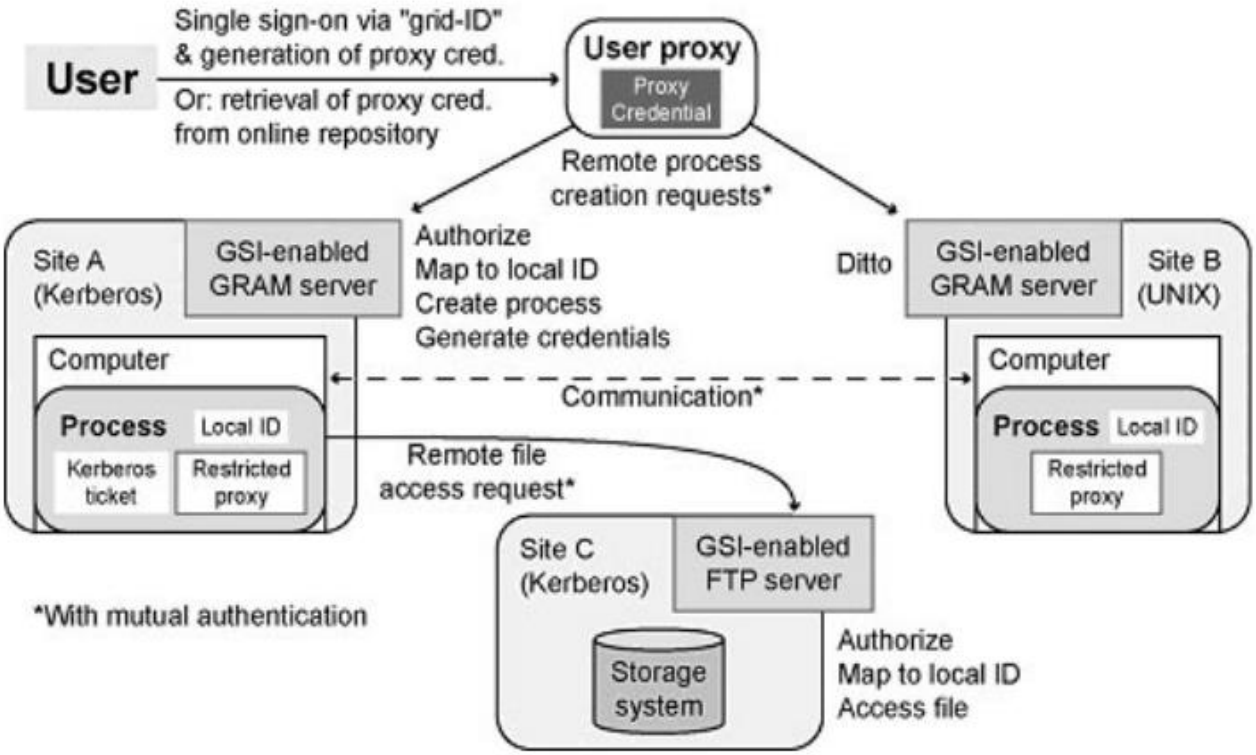


Рис.6.7 Інфраструктура безпеки

Остання версія gLite – 3.2 – була випущена в червні 2012 року, а з травня 2013 остаточно припинено підтримку проекту.

ППЗ ARC (Advanced Resource Connector)

Проект NorduGrid створений скандинавськими країнами (Данія, Норвегія, Швеція, Фінляндія) і перша його тестова експлуатація розпочалася в 2002р. Основні архітектурні рішення ARC відповідають загальноприйнятим вимогам побудови Грід. Це найпростіше ППЗ, яке підтримує всі необхідні функції для роботи Грід-інфраструктури, працює дуже стабільно і, нарешті, супроводжувалося (і супроводжується) найбільш повною і зрозумілою документацією.

В ARC реалізовані слідуючі функції:

– інформаційні (збір інформації про ресурси Грід-системи);
– динамічне підключення ресурсів в Грід -систему;
– розподілення завдань по ресурсам;
– пересилання завдань на виконання та керування цими завданнями;
– мониторинг Грід-системи.

Всі функції ARC реалізовані в вигляді окремих служб, кожна із яких основується на програмні засоби з відкритим кодом: OpenLDAP, OpenSSL, SASL. ARC – це коннектор для проекту країн Північної Європи, які

орієнтовані на підтримку додатків фізики високих енергій. Його реалізація виконана за допомогою бібліотек Globus Toolkit 2 (GT2), безпека досягається шляхом використання протоколів та інфраструктурних рішень GSI. Для опису запитів на виконання завдань в NorduGrid ARC використовується розширена мова запиту ресурсів XRSL (eXtended Resource Specification Language).

До переваг NorduGrid ARC можна віднести простоту в використанні (клієнтські компоненти пакету легко встановлюються, а серверні компоненти не вимагають реконфігурування сайту), пакет являється вільно розповсюджуваним, з відкритим вихідним кодом. Знакова особливість NorduGrid ARC – власнийй набір сервісів, які замінюють GT2. Однако багато функцій GT2 в ARC реалізуються власними засобами. Крім того, розширена мова опису ресурсів xRSL.

Слід зазначити, що українська Грід-інфраструктура використовує проміжне програмне забезпечення ARC.

UNICORE

Зараз ведеться розробка UNICORE 8.0, але остання актуальна версія UNICORE - 7.4.1, яка була випущена в 2018 році. UNICORE 7 має такі характеристики:

• клієнти:
  - веб-портал: виконання завдань, керування даними та підтримка базового робочого процесу, розширюваність;
  - графічний клієнт на основі Eclipse: Grid браузер, редактор робочих процесів, керування робочими процесами, розширюваність;
  - клієнт командного рядка: виконання робіт, передача даних, сценарії, пакетний режим, розширюваність;

• компоненти ядра сервера:
  - шлюз: маршрутизація веб-служб, обхід брандмауера;
  - UNICORE/X: Сервісний контейнер підтримує веб-служби SOAP, сервіси WSRF 1.2 і RESTful;
  - Grid сервіси: виконання завдань, доступ до сховища, метадані, передача файлів, реєстр послуг;
  - XUUDB: база даних користувачів, спільна між вузлами Grid, підтримує декілька входів і проектів;
  - гнучка система керування виконанням (XNJS): підтримка JSDL та HPC-BP, робота з розгортанням параметрів;
  - змінні механізми передачі файлів: HTTPS або ByteIO за замовчуванням, додатково UFTP, GridFTP;
  - інтерфейс цільової системи: підтримка загальних пакетних систем (LSF, LoadServer, SGE, Torque, SLURM);

• UNICORE Security:
  - на основі відкритих стандартів: X.509 Інфраструктура відкритого ключа, TLS, SAML, XACML;
  - гнучка аутентифікація користувачів: X.509, ім’я користувача/пароль, OpenID Connect;
  - багато параметрів авторизації: XUUDB, Unity файли локальних карт;

• робочий процес:
  o розширюваність: Custom resource brokering strategies;
  o служба оркестратора: брокеринг ресурсів, виконання завдань, масштабованість за допомогою розгортання декількох екземплярів;
  o потужні функції робочого процесу: графіки, петлі, умови, змінні, утримання/продовження;
• додатки:
  o автономний пакет UFTP для високопродуктивнох передачи даних (незалежно від UNICORE);
  o Unity: сервер керування ідентифікацією, сумісний з SAML, графічний інтерфейс адміністратора.

EUROPEAN MIDDLEWARE INITIATIVE

European Middleware Initiative (EMI) – результат кооперації чотирьох постачальників проміжного ПЗ для ґрід-обчислень – gLite, ARC, UNICORE та dCache. За функціональністю ППЗ розподілене на 4 основні групи сервісів: сервіси обчислення, сервіси даних, сервіси безпеки, інфраструктурні сервіси.
Сервіси обчислення включають служби ППЗ і клієнтські компоненти, які забезпечують обробку і керування запитами користувачів по виконанню обчислювальних завдань. Вони задовольняють взаємодію з локальними системами управління за допомогою єдиного інтерфейсу доступу до обчислювальних ресурсів і забезпечують доступність високорівневого метапланування, виконання потоку завдань та контроль стану завдання. Важливими є реалізація стандарту JSDL (Job Submission and Description Language, мова опису завдань в ґрід-системах) та специфікації OGF OGSA-Basic Execution Service (BES), яка планує відправлення і виконання завдань на обчислювальних ресурсах. OGSA-BES визначає операції, які мають бути реалізовані в ППЗ та необхідні для відправлення завдання на обчислення. До них входять операції з контролю стану завдання і завантаження результатів розрахунків.

Сервіси даних включають служби ППЗ і клієнтські компоненти, пов'язані з обробкою запитів користувачів з доступу до даних, зберіганням та реплікацією даних. Сервіси даних EMI повинні забезпечувати підтримку наступних стандартів:
- елементи зберігання даних мають забезпечувати POSIX доступ до даних через протокол NFS 4.1 (pNFS), дозволяючи інтегрувати розподілені системи передачі та обробки даних у простір імен локальної файлової системи;
- всі дані повинні бути доступними за допомогою веб-порталів HTTP(s) та WebDav;
- керування даними має бути забезпечено через протокол SRM;
- протоколи доступу мають бути захищені веб-стандартом аутентифікації X509/SSL, який забезпечує взаємодію із широко розповсюдженими клієнтами (веб-браузерами).

Для забезпечення стандартизації інформаційної моделі виконується впровадження підтримки всіма сервісами GLUE2-специфікації, яка описує подання інформації про ґрід-інфраструктуру.

Для системи обліку використання ресурсів зараз доступні системи APEL та SGAS, які забезпечують зберігання інформації про використання ресурсів користувачами та забезпечують публікацію цієї інформації на веб-сайті.

Проміжне програмне забезпечення Globus Toolkit, gLite, Unicore.

Попередньо ми стисло познайомилися з ***midlleware Grid.*** Так як в минулих курсах Ви мало проходили ***midlleware,*** то далі більш детально розглянемо ППЗ Globus Toolkit, ARC, Unicore, gLite.

Globus Toolkit

Управління даними

Більшість існуючих систем забезпечують передачу великої кількості даних, типи даних різні, часто передаються в режимі реального часу. Елементами системи управління даними являються:

-GridFTP - стандартний FTP протокол, розширений можливостями паралельного пересилання даних і можливостями автоматичної повторної передачі в випадку виникнення помилки.

- Reliable File Transfer - надбудова над протоколом GridFTP, яка позволє працювати з виростанням web-сервісів.

- Replica Location Service- утиліта, яка позволяє зберігати інформацію про файли чи копії файлів в системі. Представляє собою реєстр, в якому міститься інформація про фізичне розташування файлів і їх репліцірованих копій. Реєстр може бути розділеним на декілька машин в мережі, що забезпечує стійкість до збоїв обладнання.

- Data Replication Service - сервіс більш високого рівня, ніж RFT і RLS. Він об’єднує в собі ці компоненти. Задачею сервісу являється перевірка на наявність деякого набору файлів на сервері. В випадку негативної відповіді він повинен відшукати файли шляхом запиту до сервісу RLS і ініціювати передавання файлів через RFT.

Керування виконанням завдань

Архітектура керування ресурсами GRMA (Globus Resource Management Architecture) має багаторівневу структуру (рис.6.8)

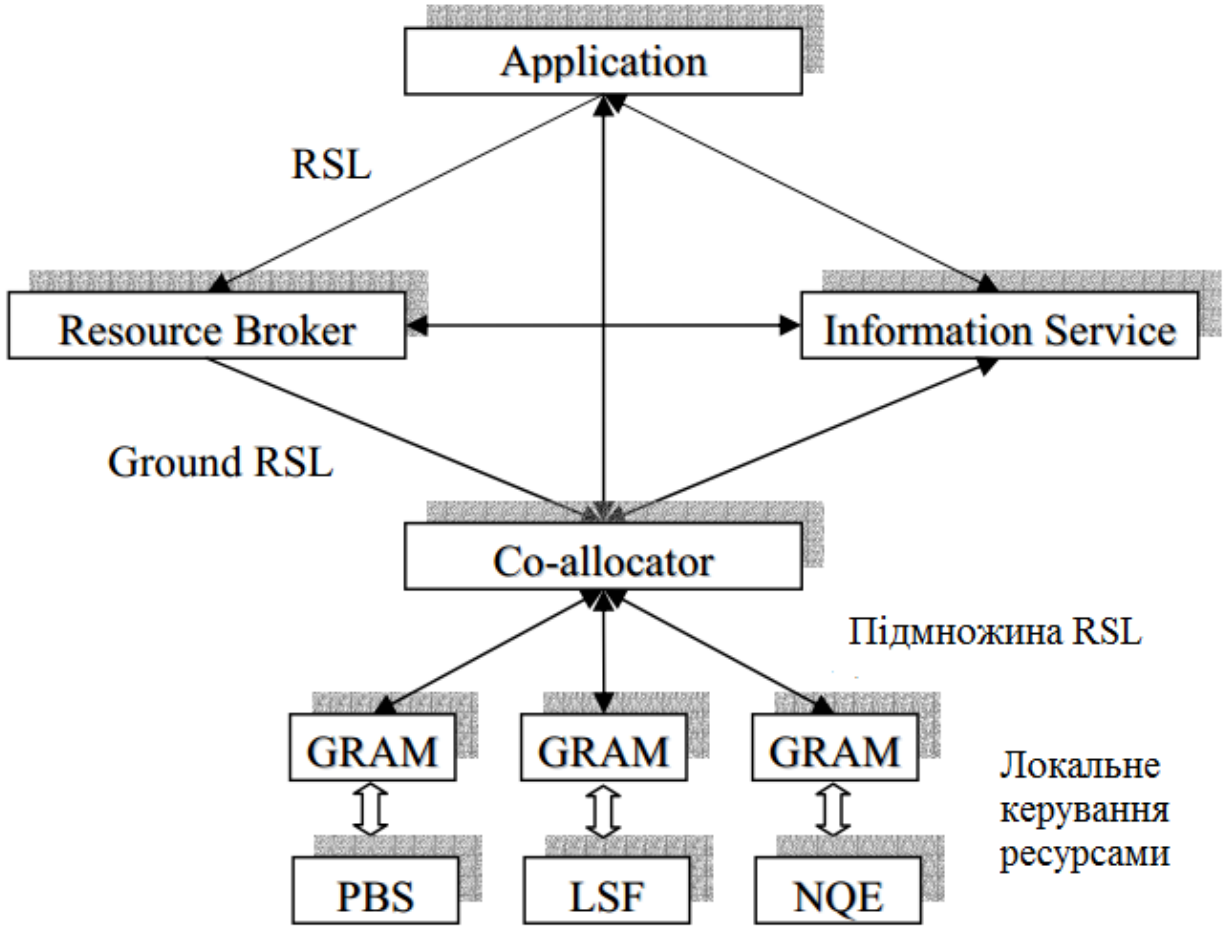


Рис.6.8 Структукра GRMA

Запити користувацьких додатків виражаються на RSL і передаються брокеру ресурсів, який відповідає за високорівневу координацію використання ресурсів (балансування навантаження) в певному домені. На основі переданого користувацьким додатком запиту та політики (права доступу, обмеження по використанню ресурсів) відповідального адміністративного домену брокер ресурсів приймає рішення про те, на яких обчислювальних вузлах буде виконуватися задача, який процент обчислювальної потужності вузла вона може використовувати та ін. При вибору обчислювального вузла брокер ресурсів має визначити, які вузли доступні в цей момент, їх загрузку, продуктивність та інші параметри, вказані в RSL-запиті, вибрати найбільш оптимальний варіант (один обчислювальний вузол, чи декілька), згенерувати новий RSL-запит (ground RSL) і передати його високорівневому менеджеру ресурсів (co-allocator). Цей запит буде містити вже більш конкретні дані, такі, як імена конкретних вузлів, необхідна кількість пам'яті та ін. Основні функції високорівневого менеджеру ресурсів GRMA перераховані нижче:

• колективне виділення ресурсів;
• збільшення/видалення ресурсів до раніше виділених;
• отримання інформації про стан задач;
• передача начальних параметрів задачам. GRMA виконує декомпозицію запитів ground RSL на множині простих RSL запитів і передає ці запити GRAM. Далі задача користувача запускається на виконання. Якщо один із GRAM повертає помилку, задача або знімається з виконання, або запускається

повторно. Високорівневий менеджер ресурсів виконує декомпозицію запитів ground RSL на множину більш простих RSL-запритів і передає ці запити GRAM. Далі, при відсутності повідомлень про помилки від GRAM, задача користівача запускається на виконання. Менеджер GRAM предоставляє верхнім рівням универсальний API для керування ресурсами вузла Грід. Сам GRAM взаємодіє з локальними засобами керування ресурсами вузла. Вузлом може бути, наприклад, робоча станція чи обчислювальний кластер.

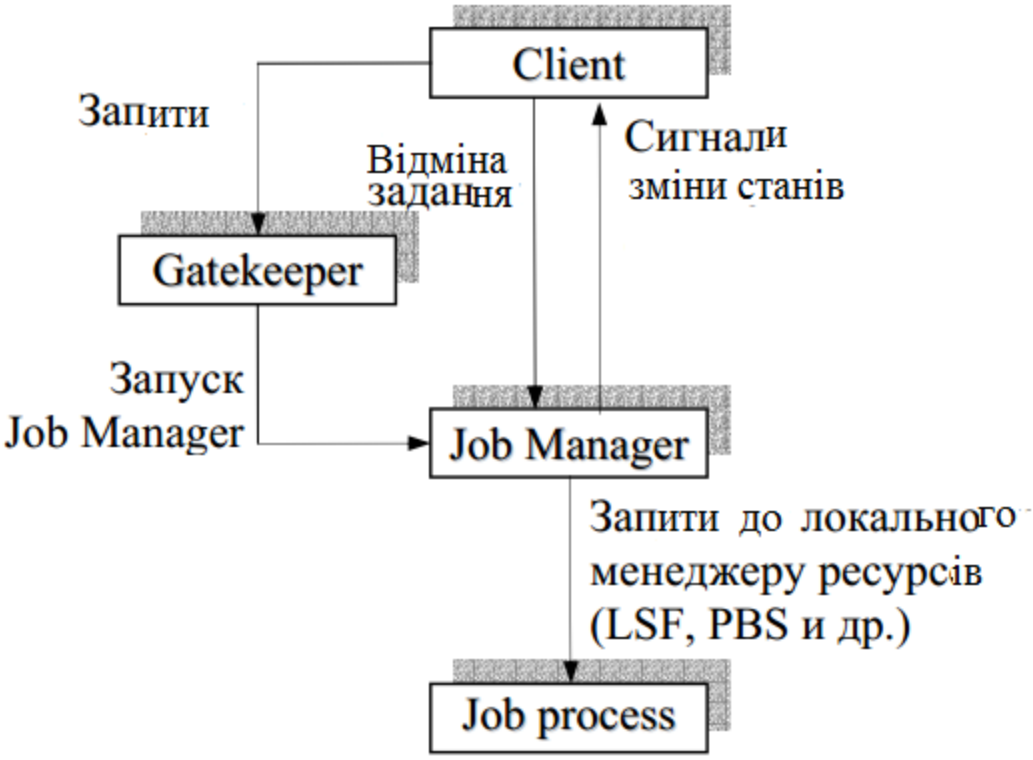


Рис 6.9 Схема взаємодії GRAM

GRAM – достатньо низькорівневий компонент Globus Toolkit, який являється інтерфейсом між високорівневим менеджером ресурсів і локальною системою керування ресурсами вузла. Цей інтерфейс може взаємодіяти з наступними локальними системами керування ресурсами:
• PBS (Portable Batch System) – система керування ресурсами і загрузкою кластерів. Може працювати на великій кількості різних платформ: Linux, FreeBSD, NetBSD, Digital Unix, Tru64, HP-UX, AIX, IRIX, Solaris. Існує вільна і володіюча більш широкими можливостями реализація PBS, називаєма Torque.
• Condor – вільно доступний менеджер ресурсів, розроблений в основному студентами різних університетів Европи і США Працює на різних платформах UNIX і Windows NT. Для того, щоб на даному обчислювальному вузлі можна було віддалено запускати на виконання програми, на ньому має виконуватися спеціальний процес, який називається Gatekeeper. Gatekeeper працює в привілейованому режимі і виконує наступні функції:
• проводить взаємну аутентифікацію з клієнтом;
• аналізує RSL запит;
• відображає клієнтський запит на облікову запись деякого локального користувача;
• запускає від імені локального користувача спеціальний процес , називаємий

Job Manager, і передає йому список потрібних ресурсів. Після того, як Gatekeeper виконає свою роботу, Job Manager запускає завдання (процес чи декілька процесів) і проводить його подальший мониторинг, повідомляючи клієнту про помилки та інші події. Gatekeeper запускає тільки один Job Manager для кожного користувача, котрий керує всіма завданнями даного користувача. Коли завдань більше не залишається, Job Manager завершує роботу.

Організація GRAM. Коротко опишемо реалізацію GRAM (рис 6.10).

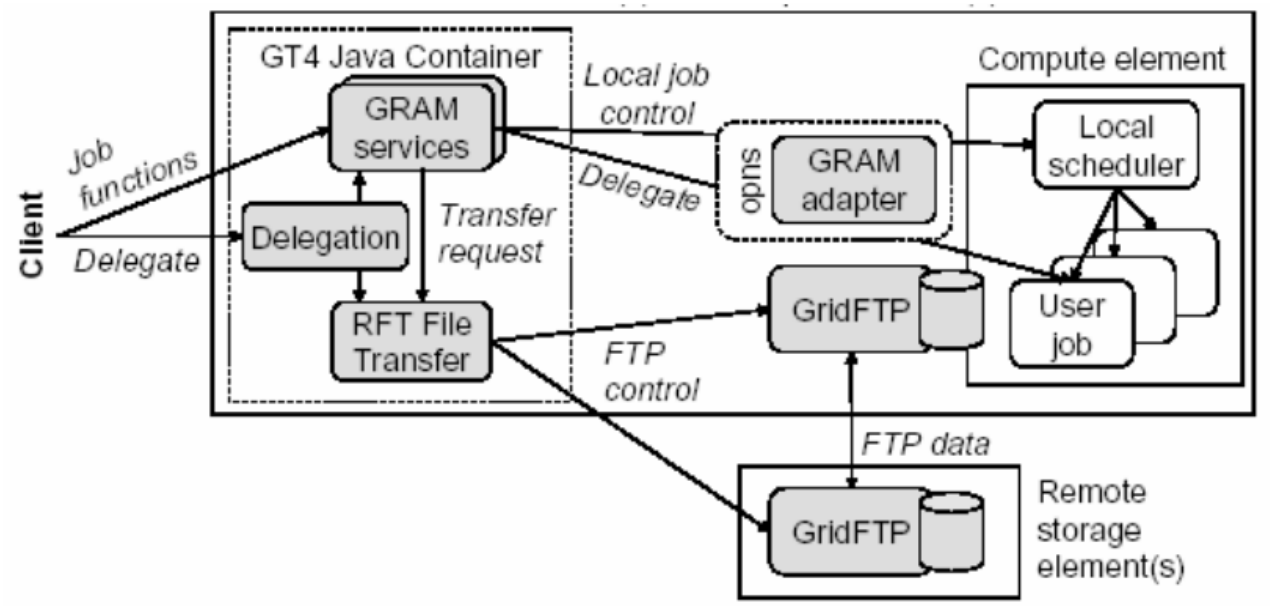


Рис 6.10 Компоненти GRAM

Основними елементами GRAM є:

• GRAM специфічні служби для створення, спостереження і керування роботами.

• Загальноцільові служби для передачі прав і надійної передачі даних RFT.

• Плануючі GRAM адаптери, предназначені для передачі запитів в відповідні локальні планувальники.

• GridFTP сервери для виконання конвейерних команд.

• Програми командного рядка (Таб.1). Інтерфейси користувача позволяють визвати компоненти безпосередньо із програми користувача, написаної на Java, C, Python.

Таб.1 - Кожний рядок представляє ряд програм

Команди	Опис
cas-*	Загальна служба авторизації
globus-credential-*	Інсталяція і відновлення прав в сужбі передачі прав
myproxy-*	Служба MyProxy
grid-ca-sign	Простий видавець для генерації X.509 прав
gsi*	OpenSSH с ssh, scp, stpf клієнтами
grid-*	Управління X.509 проксі мандатами і файлами розміщення
globus-url-copy	GridFTP клієнт
rft*	Клієнт надійної передачі файлів
globus-rls-*	Клієнт RLS, адміністрація, сервер
globusrun-ws	Клиент GRAM
mds-servicegroup-add	Добавлення входу в MDS групу служб
globus-*-container	Запуск і зупинка контейнера Globus
wsrf-*	Взаємодія з WSRF властивостями ресурсів
wsn-*	Створення і керування WS-Notification subscriptions.

В таб.2 вказані види статусної інформації, яку користувач на свій розсуд в любий момент може запросити, працюючи, наприклад, в пакетному режимі.

Таб. 2 - Стани робіт в GRAM

C	Опис
Unsubmitted	Робота ще не завдана
StageLn	Робота очікує етапу виконання чи вхідних файлів для закінчення
Pending	Локальний планувальник ще не запланував роботу на виконання
Active	Робота виконується
Suspended	Виконання роботи призупинено
StageOut	Виконання роботи завершено, вихідні файли виведені
CleanUp	Виконання роботи завершено, виконується очистка задач
Done	Робота завершена успішно
Failed	Робота потерпіла неудачу

Система Condor

Еволюція створення ППЗ привела до того, що алгоритми планування розміщення завдання на розподілених ресурсах були виділені в окрему службу, яка забезпечувала додаткові можливості по вибору ресурсів, моніторингу проходження завдань і доступу до даних з додатків. Прикладами таких програмних реалізіцій є Nimrod/G, AppLe Parameter Sweep Template, Condor-G, Gridbus Resource Broker, GridWay.

Condor – одна із багатьох спеціалізована пакетна система для керування виконанням інтенсивних комп'ютерних робіт, яка може використовуватися під GRAM. Як і інші системи подібного роду, Condor забезпечує механізм черговості, політику планування, схему пріоритетів і класифікацію ресурсів. Користувач задає комп'ютерну роботу для Condor d RSL, Condor ставить роботу в чергу, запускає її і потім інформує користувача про результати. Порядок виконання роботи такий:

Підготовка коду. Робота для Condor має бути придатною для виконання в фоновому пакетному режимі. Програма, яка виконується в фоновому режимі, не придатна для інтерактивного вводу-виводу. Condor може перенаправити консольний вивід і ввід з клавіатури в/із файла. Для цього треба створити потрібний файл, котрий містить необхідні рядки для програмного входу.
Середовище Condor. Condor має декілька средовищ для виконання (Universe): Standard, Vanilla, PVM, MPI, Globus, Java, Scheduler. Користувач має вибрати одне із них.
Задання файлу опису. Цей файл містить такі дані, як ім'я програми, яка буде виконуватися, файли для використання даних екрану і клавіатури, тип платформи, необхідної для виконуємої програми і багато іншої інформації.
Задання роботи. Робота для Condor задається командою *condor_submit*. Після цього Condor самостійно виконує дії по запуску і виконанню завдання.

Інформаційний сервіс

Основним елементом системи інформаційного сервісу являється підсистема оновлення і відшукування сервісів MDS (Monitoring and Discovery Service), в яких зареєстровані всі ресурси Грід. Інформація про ресурси може містити, наприклад, як дані про конфігурацію чи стану як всієї системи, так і окремих її ресурсів (тип ресурсу, доступний дисковий простір, кількість процесорів, об'єм пам'яті, продуктивність та інше). Вся інформація логічно організована в вигляді дерева і доступ до неї проводиться по стандартному протоколу LDAP (Lightweight Directory Access Protocol). Кожна запись в структурі LDAP містить властивості окремого ресурсу. Користувацький додаток може запросити сервер для встановлення типу архітектури, підтримуваємої операційною системою, даних про менеджер задач, які доступні на заданому ресурсі. Кожний організаційний домен, який входить в Грід, має запустити сервіс інформації про ресурси (GRIS). Функціональність даного сервісу включає в себе:

• Огляд і отримання інформації, яка може бути точно співставлена з конкретним ресурсом чи об'єктом.
• Можливість робити запити і пошук інформації для отримання набору зв'язаних ресурсів чи об'єктів.
• Створення нової інформації по запиту, яка не збережена в системі.
• Переадресація подій для передачі інформації, основаної на динамічних подіях в середині інфраструктури Grid.
• Збір інформації для тематичного отримання і організації інформації.
• Фільтрація інформації для зменшення її кількості і збільшення продуктивності.
• Зберігання, резервне копіювання, кешування інформації для більш простого наступного доступу од неї і для підвищення продуктивності.
• Безпека, захист і шифрування для забезпечення роботи важливих задач, таких як контроль доступу, аутентифікація.
• Для підтримки даної функціональності користувачами Грід-системи можуть бути використані кілька сервісів. MDS має децентралізовану,

легкомасштабовану структуру і працює як зі статичними, так і з динамічними даними, необхідними користувацьким додаткам і різним сервісам Грід-системи. Ієрархічна структура MDS представлена на рис.6.11.

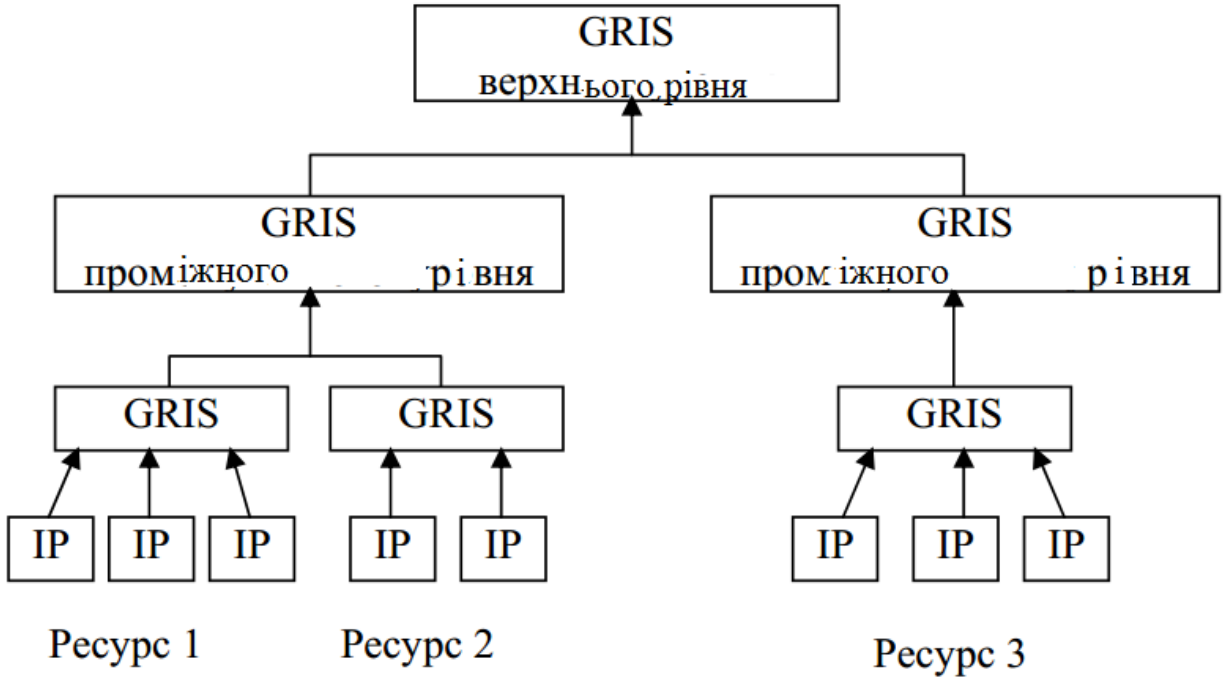


Рис.6.11 Ієрархічна структура MDS

MDS складається із трьох основних компонент:

1. IP (Information Provider) – являється джерелом інформації про конкретний ресурс.

2. GRIS (Grid Resource Information Service) – предоставляє інформацію про вузол Грід-системи, який може бути як обчислювальним вузлом, так і будь-яким іншим ресурсом. GRIS опитує індивідуальні IP і об'єднує отриману від них інформацію в рамках єдиної інформаційної схеми.

Пакет gLite

gLite - європейський пакет middleware для Грід.

Нижче перераховані основні елементи і їх призначення:

CE (Computing Element) - набір програм для установки на керуючий вузол обчислювального кластеру. Даний елемент надає універсальний інтерфейс до системи управління ресурсами кластеру і дозволяє запускати на кластері обчислювальні завдання;

SE (Storage Element) - набір програм, призначений для установки на вузол зберігання даних. Даний елемент надає універсальний інтерфейс до системи зберігання даних і дозволяє керувати даними (файлами) в Грід-системі;

WN (Worker Node) - набір програм, призначений для установки на кожний обчислювальний вузол кластеру. Даний елемент надає стандартні функції і бібліотеки LCG завданням, які виконуються на даному обчислювальному вузлі; (LCG – це Large Hadron Collider Computing Grid, предназнач. для великого адронного колайдера)

UI (User Interface) - набір програм, що реалізують інтерфейс користувача Грід-системи (інтерфейс командного рядка). В цей елемент входять стандартні команди керування завданнями і даними, деякі з яких розглянуті далі;

RB (Resource Broker) - набір програм, які реалізують систему керування завантаженням (брокер ресурсів). Це найбільш складний (і об'ємний) елемент LCG, що надає всі необхідні функції для скоординованого автоматичного керування завданнями в Грід- системі;

PX (Proxy) - набір програм, які реалізують сервіс автоматичного оновлення сертіфікатів (myproxy);

LFC (Local File Catalog) - набір програм, які реалізують файловий каталог Грід-стеми. Файловий каталог необхідний для зберігання інформації про копії (репліки) файлів, а також для пошуку ресурсів, які містять необхідні дані;

BDII (Infornation Index) - набір програм, які реалізують інформаційний індекс Грід-системи. Інформаційний індекс містить всю інформацію про поточний стан ресурсів, отримувану з інформаційних сервісів, і необхідну для пошуку ресурсів;

MON (Monitor) - набір програм для моніторингу обчислювального кластеру. Даний елемент збирає і зберігає в базі даних інформацію про стан і використання ресурсів кластеру.

VOMS (VO Management Service) - набір програм, які реалізують каталог віртуальних організацій. Даний каталог необхідний для управління доступом користувачів до ресурсів Грід-системи на основі членства в віртуальних організаціях.

До базових систем gLite відносяться такі системи:

• **Система управління завданнями WMS** (Workload System). Завданням підсистеми управління загрузкою є прийняття запитів на запуск завдань, пошук відповідних ресурсів і контроль їх виконання. Завдяки роботі WMS, складність управління додатками і ресурсами в Гріді прихована від користувачів. Їх взаємодія з WMS обмежена описом характеристик і вимог запиту через орієнтовану на користувача мову опису завдань (Job Description Language, JDL) і до направлення такого запиту через надані інтерфейси.

• **Система керування даними**. Прикладне завдання повинно бути в змозі звернутися до своїх даних, незалежно від фактичного місця розташування обчислювального ресурсу. Найбільш поширеним таким інтерфейсом є Менеджер ресурсів зберігання даних (Storage Resource Manager, SRM).

• **Система інформаційного обслуговування і моніторингу** Грід-системи вирішує завдання збору і управління даними про стан Грід, отримуючи інформацію від безлічі розподілених джерел - постачальників. Підсистема призначена для постійного контролю функціонування Грід і забезпечення своєчасного реагування на виникаючі проблеми.

• **Система безпеки і контролю прав доступу**. Щоб Грід був ефективною структурою для розподілених обчислень, призначені процеси і служби Грід для користувача повинні працювати в безпечному середовищі. Для цього взаємодії між компонентами Грід повинні бути взаємно аутентифіковані (аутентифікація - процес підтвердження заявлених властивостей об'єкту на підставі засвідчень його документів); будь-яка дія повинна відбуватися тільки

після відповідної авторизації - зіставлення об'єкту, який здійснює дію, і набору прав, наданих цьому об'єкту для роботи в Грід-середовищі.

• **Система протоколювання**. Підсистема протоколювання відстежує процес виконання завдань, здійснюваний під управлінням WMS. Вона збирає сповіщення про події від різних компонентів WMS і обробляє їх, щоб представити узагальнений поточний стан (статус) завдання.

• **Система обліку**. Ця підсистема призначена для обліку використання обчислювальних ресурсів (таких як процесорний час, використання оперативної пам'яті і так далі), і, зокрема, може використовуватися для формування інформації про вартість використання даного Грід-ресурсу даним користувачем, якщо взаємини користувачів і провайдерів ресурсів засновані на економічній моделі.

Управління виконанням завдань

Основна і найбільш розвинена частина в складі комплексу gLite - це система управління завданнями (Workload Management System, WMS). Її призначення - підтримка виконання програм на розподілених і організованих у вигляді гріда комп'ютерах

Для користувача Грід обчислювальна діяльність виглядає таким чином: він запускає завдання, яке доставляється на один, або в разі багатопроцесорних завдань - на кілька комп'ютерів із загального ресурсного пулу Грід; завдання виконується, а результати можуть бути отримані на робоче місце, з якого здійснювався запуск.

Реалізовані в WMS технології підтримують розподілену обробку такого роду, забезпечуючи:

- автоматичне виділення відповідних виконавчих комп'ютерів (це основна відмінність WMS від базових засобів керування завданнями, в яких потрібна явна вказівка виконавчих ресурсів);
- переміщення програми, вхідних і діагностичних файлів;
- створення середовища виконання, в тім рахунку домашньої директорії, на виконавчому комп'ютері.

Форми користувацької діяльності в Грід і на окремому комп'ютері відрізняються в декількох аспектах:

• Програмний код завдання виконується на виконавчому комп'ютері без участі користувача (в пакетному режимі). Сам код не вимагає адаптації до умов Грід. В деяких випадках може знадобитися його доповнення прологом / епілогом, які виконують підготовчі / завершальні операції, наприклад, доставку оброблюваних даних.

• Завдання представляється WMS у вигляді формалізованого опису, складеного на мові JDL (Job Description Language).

• Ресурси Грід використовуються колективно безліччю користувачів, тому Завдання не обов’яково починає виконуватися відразу після запуску: воно може

чекати звільнення ресурсів, зайнятих іншими завданнями. Очікуючі ресурсів завдання зберігаються в чергах (WMS або ресурсних центрів CE).
• Питання безпеки в Грід gLite вирішуються на основі принципів GSI. Перед тим як почати працювати з WMS (або будь-якою іншою системою gLite), користувач повинен згенерувати тимчасовий проксі-сертифікат за допомогою команд grid-proxy-init або voms-proxy-init.

Компоненти системи керування виконанням завдань

Після відправки за допомогою інтерфейсу користувача запит обробляється декількома компонентами WMS. Їх точний склад може варіюватися в залежностісті від конкретного ППЗ, але зазвичай в нього входять:
• менеджер завантаження (можливо з додатковим проксі-сервером);
• планувальник / брокер ресурсів;
• адаптер завдань;
• модуль, відповідальний за виконання фактичних операцій з керування завданнями (напрямок на виконання, видалення завдання, і т.д.);
• монітор log-файлів WMS.

Менеджер завантаження - основний компонент WMS. Після отримання запиту, він повинен вжити відповідних заходів, щоб виконати його. Щоб зробити це, він взаємодіє з іншими компонентами WMS, набір яких залежить від типу запиту. Для підвищення продуктивності системи (кількості запитів, які можуть бути оброблені за одиницю часу), цей компонент доповнюється проксі-менеджером завантаження, відповідальним за прийом поступаючих запитів від інтерфейсу користувача (наприклад, на запуск або видалення завдання), аналіз їх коректності та, якщо результат аналізу позитивний, передачу запиту менеджеру завантаження.

Основним типом запитів до WMS являються запити на виконання завдань. А головними результатами їх обробки підсистемою являються:
• правильний підбір грід-ресурсу для виконання конкретного завдання на підставі опису завдання на мові JDL та інформації про доступні ресурси.
• направлення завдання на обраний ресурс.

Першу задачу вирішує компонент, який називається планувальником або брокером ресурсів (Resource Broker, RB). Існування відповідних ресурсів для даного завдання залежить не тільки від стану ресурсів, але також і від політики їх використання, якою слідують адміністратори ресурсу і / або адміністратор віртуальної організації, до якої належить користувач.

Брокер ресурсів може слідувати різній стратегії при розподілі завдань по ресурсам:
• один крайній варіант стратегії - після надходження запиту на виконання завдання, як можна швидше підбираються найбільш підходящі ресурси та, як тільки рішення прийнято, завдання передається на обраний ресурс для виконання («нетерпляче планування», в англомовній літературі називається також push-model);
• інший крайній варіант - завдання знаходяться в менеджері завантаження, поки будь-який ресурс не стає доступним, після чого вивільненому ресурсу

підбирається найбільш відповідне завдання (що чекали) і передається цьому ресурсу для виконання («лінива політика планування», pull-model);
• можливі проміжні комбіновані варіанти стратегії підбору ресурсів.

Для виконання своїх завдань брокер використовує базу даних про ресурси, яка оновлюється внаслідок або отримання повідомлень від ресурсів, або активного опитування ресурсів системою інформаційного обслуговування і моніторингу, або деякої комбінації цих двох механізмів оновлення. Крім того, спеціальний компонент WMS, який відповідає за взаємодію з інформаційною підсистемою, може бути налаштований так, щоб певні повідомлення про зміну станів ресурсів могли викликати початок роботи брокера. Іншим важливим компонентом, що підвищує стійкість роботи WMS, є модуль керування чергою завдань, який дає можливість деякий час зберігати запит на виконання завдання, навіть якщо ніякі ресурси, відповідні його вимогам, не доступні негайно. Запити, по яким не вдалося відразу підібрати ресурси, будуть повторені або пеперіодично (при «нетерплячому плануванні»), або як тільки повідомлення про нові доступні ресурси з'являються в сховищі даних (при «лінивій політиці планування»). В протилежному випадку, такі ситуації могли б привести до негайного припинення виконання завдання через відсутність відповідного ресурсу.

Далі **слідує модуль, відповідальний за виконання фактичних операцій** з керування завданнями (направлення на виконання, видалення завдання, і т.д.), які виконуються за запитами менеджера завантаження. У gLite в якості цього модуля використовується Condor C.

Монітор log-файлів відповідальний за перегляд повідомлень модуля керування завданнями і перехоплення подій про зміну станів завдання.

Опис завдання

Користувач взаємодіє з WMS зі свого робочого місця, на якому встановлено інтерфейс (User Interface, UI) цієї системи, що включає наступний набір команд:
glite-job-submit <jdl_file> запуск завдання, опис якого міститься в файлі <jdl_file
glite-job-cancel <job_Id> зняття завдання в будь-який момент процесу обробки.
glite-job-status <job_Id> отримання стану завдання.
glite-job-output <job_Id> отримання сумарної діагностики від всіх х етапів обробки.

Як видно з цього списку, набір команд досить повний: він дозволяє запустити завдання, відслідковувати хід обробки і при необхідності зняти його. Повертаючись до особливостей роботи в Грід, звернемо увагу на те, що обробка команд проводиться розподіленим чином різними службами WLMS, які взаємодіють між собою по мережі. У зв'язку з цим час виконання команд досить великий, близько хвилини.

Файл опису завдання - JDL-файл задається як параметр команди job-submit і містить характеристики завдання у вигляді пар: <атрибут> = <значення>.
(наприклад)

```
[Type = "Job";

JobType = "Normal";

Executable = "myexe";

StdInput = "myinput.txt";

StdOutput = "message.txt";

StdError = "error.txt";

InputSandbox = {"/users/pacini/example/myinput.txt",
"/users/pacini/example/myexe"};

OutputSandbox = {"message.txt", "error.txt"};

Requirements = other.GlueCEInfoLRMSType == "PBS";
Rank = other.FreeCPUs;]
```

Серед атрибутів можна виділити три основні групи, які визначають: тип завдання, використовувані в завданні файли і спосіб вибору ресурсів.

Тип завдання описується двома атрибутами: Type і JobType. WMS підтримує роботу як з простими завданнями, так і з складовими, атрибут Type має, відповідно, значення "Job" або
"DAG". Тип DAG (direct acyclic graph of dependent jobs) відповідає завданням, в якому повинні бути виконані в певній послідовності ряд простих завдань.

Для простого завдання застосуємо другий атрибут - JobType. Воно може бути звичайним ("Normal"), а також:

• інтерактивним ("Interactive"). Завдання такого роду підтримує зв'язок з точкою запуску

• паралельним (MPICH), тобто вимагає для виконання декількох процесорів і ін.

Прості завдання

WMS забезпечує переміщення файлів між комп'ютером робочого місця і виконавчим комп'ютером, але обмежується, проте, мінімально необхідним набором стандартних файлів операційної системи (Unix): виконуваним файлом завдання, вхідних даних, діагностичних повідомлень і файлом повідомлень про помилки.

Вхідні файли (виконуємий і файл даних), які лежать в файловій системі робочого комп'ютера користувача UI, передаються спочатку на сервер WMS, а потім при запуску завдання на виконавчому комп'ютері завантажуються в створену для цього завдання домашню директорію.

Вихідні файли діагностики і помилок породжуються при виконанні завдання в домашній директорії виконавчого комп'ютера, і доставляються по його закінченні не в точку запуску, а на сервер WMS, звідки їх можна отримати

командою glite-job-output. Аналогічно вхідним файлам, вихідні повинні бути описані в атрибутах OutputSandbox, StdOutput і StdError.

Оскільки переміщення файлів відбувається через тимчасовий буфер на сервері WMS, передбачається, що всі вони будуть невеликого розміру. Тим самим, ці засоби доставки файлів спрямовані на мінімально необхідну підтримку віддаленого виконання програм.

Що стосується прикладних даних, які обробляються або виробляються додатком, то для роботи з ними у програмі повинні використовуватися засоби системи **керування даними**.

Одне з найважливіших досягнень WMS - технологія віртуалізації, яка реалізує автоматичний розподіл завдань на виконавчих ресурсах. Розподіл проводиться з точністю до ресурсного центру, тобто визначається не конкретний виконавчий комп'ютер, а деякий CE. При цьому в певній мірі враховується стан і склад його ресурсів (число завдань в черзі, платформа виконавчих комп'ютерів і т.д.). Ці відомості представляються диспетчеру завдань WMS інформаційною службою gLite.

Схема обробки завдань

Наступний рис.6.12 ілюструє процес виконання в Грід деякого простого завдання.

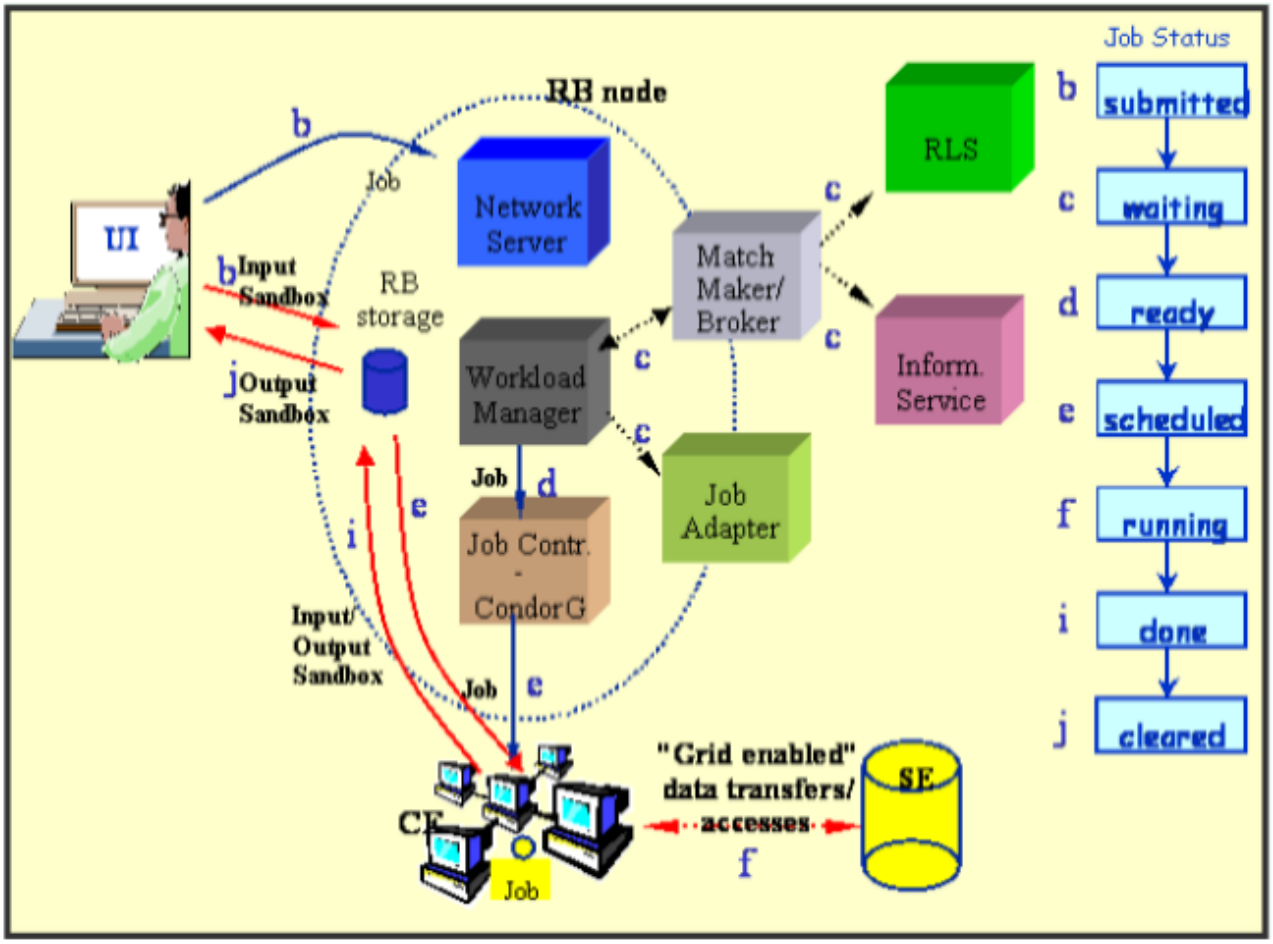


Рис.6.12 Виконання завдання в gLite. Справа представлені стани виконання завдання

1. Після отримання цифрового сертифікату від Certification Authority, реєстрації в ВО (Віртуальній Організації), користувач може використовувати gLite. Він реєструється в UI і створює проксі-сертифікат, щоб представляти себе в операціях з безпеки.
2. Користувач запитує роботу через UI до Resource Broker. В описі роботи вказані один або більше файлів, які потрібно скопіювати в RB. Цей ряд файлів називається Input Sandbox. Подія реєструється в LB і роботі присвоюється статус **запрошено**.
3. WMS шукає найкращий доступний CE для виконання роботи. Щоб зробити це, він запитує **Information Supermarket (ISM**, внутрішній кеш інформації, який в цій системі читає з BDII, щоб визначити статус обчислювальних ресурсів і ресурсів пам'яти і в File Catalogue, щоб знайти місце будь-якого необхідного вхідного файлу. Інша подія реєструється в LB і встановлюється статус роботи ОЧІКУВАННЯ.
4. RB готує роботу для запиту виконання, створюючи оболонку скрипту, який буде пересланий разом з іншими параметрами в обраний CE. Подія реєструється в LB і роботі присвоюється статус ГОТОВИЙ.
5. CE отримує запит і посилає роботу на виконання в локальний LRMS.
6. Подія реєструється в LB і роботі присвоюється статус СПЛНОВАНО.
7. LRMS керує виконанням роботи на Worker Nodes. Файли вхідного Sandbox копіюються з RB в доступний WN, де робота виконується. Подія реєструється в LB і роботі присвоюється статус ВИКОНАННЯ.
8. Під час виконання роботи файли Грід можуть бути доступні з SE, використовуючи або RFIO, або протокол перекриття, або після копіювання їх в локальну файлову систему на WN за допомогою Data Management.
9. Робота може створювати нові вихідні файли, які можуть бути перезавантаженими в Грід і стати доступними для інших користувачів. Це може бути досягнуто за допомогою Data Management. Перезавантаження файлу означає копіювання його в Storage Element і реєстрацію в каталозі файлів.
10. Подія реєструється в LB і роботі присвоюється статус ЗРОБЛЕНО.
11. З цієї точки користувач може отримувати вихідний файл його роботи в UI, подія реєструється в LB і роботі присвоюється статус **ОЧИЩЕНО**.
12. Запити на статуси роботи можуть бути адресовані LB з UI. Отже, з UI можливо робити черги BDII для статусів ресурсів.
13. Якщо сайт, до якого надіслана робота, нездатний прийняти, або виконати роботу, робота може бути автоматично і багаторазово переадресовано іншим відповідним CE. Користувач може отримати інформацію про історію роботи, запитуючи службу LB.

Система **UNICORE**

Проект UNICORE (Uniform Interface to Computing Resources - єдиний інтерфейс до обчислювальних ресурсів) зародився в 1997 році і до цього моменту являє собою комплексне рішення, орієнтоване на забезпечення прозорого безпечного доступу до ресурсів Грід.

Архітектура складається з чотирьох рівнів:

1. Клієнтській рівень.

2. Шлюз.
3. Серверний рівень.
4. Системний рівень.

Клієнтська частина складеться з:

- UCC (Unicore Command Line Client) – клієнт командного рядка для UNICORE. Забезпечує постановку задачі і вивід отриманих результатів. Підтримує Json для опису задач, вміє працювати в фоновому режимі і працювати зі скриптами на Groove.
- URC (Unicore Rich Client) – багатофункціональний клієнт UNICORE. Заснований на Eclipce. Забезпечує перегляд ресурсів Грід-сітки. Дозволяє детально описати задачі і описати потоки задач. Дозволяє керувати безпекою і моніторити виконання потоку задач.
- HiLA (High Level API for Grid Applications) - високорівневий програмний інтерфейс для додатків грід. Є підтримка Java.
- Портали – надає доступ користувачам до Грід- ресурсів через інтернет.

Шлюз слугує компонентом, який забезпечує доступ до вузла UNICORE, через автентифікацію всіх вхідних повідомлень, використовуючи стандарт X.509.

Серверний рівень є проміжним шаром, який включає в себе всі сервіси і компоненти UNICORE, заснованих на стандартах WSRF і SOAP. Включає в себе реєстр, який забезпечує реєстрацію і пошук ресурсів, доступних в Грід-середовищі. Компонент XNJS (eXecution Network Job Supervisor) забезпечує керування завданнями і виконання ядра UNICORE. Системний рівень являє собою інтерфейс цільової системи (TSI – Target System Interface), який забезпечує взаємодію між UNICORE і окремими ресурсами Грід-сітки.

Слід зазначити, що проект UNICORE був ініційований для німецьких суперкомп'ютерних центрів в якості альтернативи для Globus Toolkit. Основи виробничої версії були закладені в продовжені проекту Unicore Plus, який закінчився в 2002 р. Наступні європейські проекти розширили функціональні можливості і були направлені на забезпечення реалізації стандартів Open Grid Forum. Це привело до выпуску UNICORE 6 28 серпня 2007 р. Зараз ведеться розробка UNICORE 8.0, але остання актуальна версія UNICORE - 7.4.1, яка була випущена в 2018 році. Архітектура поточної версії UNICORE складається з трьох рівнів: клієнтського, серверного та рівня цільової системи.

Клієнтський рівень представлений наступними додатками, які входять до складу UNICORE:

- Portal, що дозволяє користувачам отримати доступ до обчислювальних ресурсів та даних з використанням браузера без необхідності встановлювати додаткове програмне забезпечення на користувацький комп'ютер;
- URC (UNICORE Rich Client) – графічний клієнт, побудований на Eclipse Framework, для персональних комп'ютерів;
- UCC (UNICORE command line client) – клієнт для командного рядка з можливістю розширення набору команд скриптами мовою Groovy.

Також можливе створення користувацьких прикладних програм безпосередньо з використанням клієнтських інтерфейсів прикладного програмування мовою Java.
Серверний рівень представлений сервісами та компонентами, побудованими на стандартах WSRF 1.2, SOAP та WS-I:
- Gateway (шлюз) – зворотній HTTPS проксі-сервер, який виконує роль мережевого екрана та точки входу до порталу UNICORE, переадресовуючи запити підключених клієнтів на сервери за ним;
- UNICORE/X – центральний компонент UNICORE, він приймає запити клієнтів, які пройшли через шлюз, виконує їх автентифікацію, перевірку авторизації та викликає відповідний сервіс. Він надає набір інтерфейсів сервісів, що розміщуються в контейнері UNICORE Services Environment (USE) та виконують управління завданнями, передачу файлів, надають доступ до сховища, тощо. Сервер UNICORE/X приймає завдання на виконання, передає їх на локальні системи через TSI, дозволяє завантаження та вивантаження даних з використанням кількох протоколів. UNICORE/X може для зберігання даних використовувати такі ресурси, як сховище CDMI, S3 чи Apache Hadoop HDFS;
- Registry (реєстр) – сервер аналогічний до UNICORE/X, проте виконується як окремий сервіс в оточенні WSRF та дозволяє клієнтам виявляти доступні сервіси, розміщені на декількох серверах UNICORE/X;
- Workflow Engine та Service Orchestrator – компоненти, що відповідають за управління виконанням завдань в мережі Grid.
Рівень цільової системи (TSI) складається з інтерфейсів локальної операційної системи, файлової системи та системи управління ресурсами. Даний рівень використовується для виконання завдань, операцій з файловим введенням/виведенням та перевірки стану завдання.

Сумісніять проміжного програмного забезпечення Грід. European Middleware Initiative (EMI)

З попередніх лекцій Ви маєте знати, що Грід забезпечує вирішення складних завдань шляхом інтеграції розподілених ресурсів, ідентифікації користувачів, автоматичного виконання їх запитів і пошуку для них потрібних ресурсів. Базові функції Грід реалізує за допомогою спеціального програмного забезпечення, яке має назву ***проміжного програмного забезпечення (ППЗ),*** або ***midlleware.*** ППЗ використовується для того, щоб приховати різнорідну природу Грід та створити прозорий доступ до прикладного програмного забезпечення. ППЗ дозволяє створити однорідне середовище шляхом використання стандартизованих інтерфейсів і цілого ряду сервісів. ППЗ перебуває між операційною системою і додатками.

Як і на початку появи перших персональних комп’ютерів кожна фірма окремо створювала власні ПК і про їх стандартизацію ніякої мови не було, так і створення різних Грід (як національних так і регіональних) супроводжувалося розробкою власного проміжного програмного забезпечення : ARC, Alien, LCG, Unicore, gLite, Globus Toolkit та ін. (прогляньте попередні лекції) Всі вони розрізняються між собою, але мають багато спільного,

оскільки використовують, як правило, систему Globus Toolkit. Архітектура і програмна реалізація такого ППЗ різна і дослідницькі проекти обмежені можливостями однієї Грід-інфраструктури через відмінності в застосованому ППЗ. Але відмінність в ППЗ не єдина проблема. Використання віртуальних організацій (ВО) за галузевим напрямком (фізикам потрібно одне, а біологам інше) стало логічним продовженням розвитку Грід-комп'ютингу, проте ВО виявилися організаційно обмеженими, так як структури, які входять до їх складу, мають свої специфічні механізми доступу до ресурсів і процедури використання Грід, які визначаються різним ППЗ. Побудова і функціонування ВО як об'єднання ресурсів різних Грід, стала ще однією складністтю на шляху забезпечення функціональної сумісності.[1]

Забезпечення взаємодії Грід, які працюють під управлінням різного ППЗ, є актуальною задачею.

Функціональна сумісність (Interoperaility) визначається як здатність обмінюватися інформацією і здатність використовувати ту інформацію, якою обмінювалися.

Взаємодія (Interoperation) визначається як використання систем, які мають можливість взаємодіяти.

Грід-функціональна сумісність – це здатність ППЗ працювати спільно, а Грід-взаємодія – це здатність Грід-інфраструктур працювати спільно. Функціональна сумісність – це перший крок в напрямку до взаємодії Грід-систем. Правда, це з точки зору користувача Грід-системи, бо він повинен мати можливість виконати складне завдання, яке вимагає великих обчислювальних ресурсів, відправивши його на виконання в потрібний Грід, і отримати результат.

Моделі забезпечення функціональної сумісності можна поділити на три групи:

- Вирішення проблеми функціональної сумісності без зміни ППЗ. До цієї групи відносяться сценарії користувача і паралельне використання Грід.
- Моделі проміжної стадії. До цієї групи відносяться розробка адаптерів, трансляторів, шлюзів.
- Моделі, які забезпечують універсальне вирішення проблеми функціональної сумісності: розробка універсального інтерфейсу для доступу до Грід-систем, розробка єдиного стандарту Грід.

Сценарій користувача

В межах віртуальної організації, побудованої на основі різних Грід з різним ППЗ, можна досягти функціональної сумісності шляхом їх окремого використання. Для цього необхідно на персональному комп'ютері встановити і конфігурувати грід-орієнтований користувацький інтерфейс для кожної із цих систем, зареєструватися в ВО, які забезпечують доступ до різних Грід-інфраструктур і таким чином можна працювати з цими Грід. Але користувач сам повинен розподілити обчислення між Грід-інфраструктурами, відправляючи завдання на виконання в конкретну Грід залежно від поточного її завантаження. А ще потрібно мати різний опис одного і того ж прикладного завдання для кожної з Грід, які використовує ВО. Для віртуальної організації

необхідна реєстрація в різних Грід, які вона повинна використовувати. При цьому недостатньо використовуються можливості Грід і проблематично локалізувати збої при виконанні завдань.

Паралельне використання

В межах організації, яка має потужний обчислювальний центр, отримати функціональну сумісність можна встановленням інтерфейсу доступу до різних Грід на окремому виділеному сервері, який входить до складу обчислювального кластеру. При цьому обчислювальний ресурс буде частиною різних Грід-інфраструктур і доступ до кожної з них буде шляхом використання встановлених Грід-сервісів. Для такої роботи користувач має бути зареєстрованим в кількох Грід, відправляючи завдання на виконання в ту чи іншу систему залежно від її завантаження. Тут так само потрібно мати різний опис прикладного завдання для кожної Грід, яку використовує ВО. Відмінність від попередньої моделі в тому, що доступ у різних Грід здійснюється через один ресурс, який має бути зареєстрованим в різних Грід. Основний недолік – адміністратор системи має стати експертом в питаннях інсталяції, конфігурування та підтримки працездатності сервісів різного ППЗ.

Шлюз

Шлюз (Gateway) - це специфічний сервіс, який забезпечує єдиний доступ до ресурсів грід-інфраструктур, які працюють під керуванням різного ППЗ, і робить різні грід схожими на єдиний обчислювальний ресурс. Для забезпечення функціональної сумісності ППЗ розроблюється окремий модуль – шлюз або Gateway, що забезпечує переклад завдання користувача, написаного на мові опису завдання для однієї Грід у формат мови опису завдання для іншої Грід-інфраструктури.

З точки зору користувача є один опис завдання для Грід, а переклад його на мову опису завдання, придатну для використання в іншій Грід-інфраструктурі виконується Gateway-модулем. Для такої системи характерно:

- кожна грід працює незалежно під керуванням власного ППЗ;
- функції по узгодженню форматів мови опису завдання, авторизації і аутентифікації користувача, передачі файлів завдання і даних на обчислювальні ресурси і відправлення результатів обчислення користувачеві реалізуються в окремому модулі;
- модуль Gateway є єдиною точкою обміну пакетами між Грід і користувачем.

Використання модулів Gateway є вузьким місцем з погляду універсальності, але такий підхід корисний як доказ концепції і демонстрації реального досягнення функціональної сумісномті. Цей метод був використаний в EGEE (*Enablig Grid for E-sciensE)* для забезпечення функціональної сумісності грід-інфраструктур UNICODE і OSG. (Перша промислова грід була побудована в 2004р в проекті EGEE).

Адаптери і транслятори

Цей метод забезпечення функціональної сумісності заснований на використанні адаптерів (adaptpor) і трансляторів, які є частиною інтнрфейсу користувача Грід.

Еволюція створення ППЗ привела до того, що алгоритми планування розміщення завдання на розподілених ресурсах були виділені в окрему службу, яка забезпечувала додаткові можливості по вибору ресурсів, моніторингу проходження завдань і доступу до даних з додатків. Такий проміжний шар названо Ресурс Брокер. Прикладами таких програмних реалізіцій є Nimrod/G, AppLe Parameter Sweep Template, Condor-G, Gridbus Resource Broker, GridWay. До них можна віднести і планувальники завдань, такі як Application Level Schedultrs.

Фактично користувач має один вхід в різні Грід-системи, однаково готує завдання на виконання і визначає необхідні для його виконання ресурси. Використання адаптерів дозволяє підключати декілька Грід, а трансляторів – змінювати інформацію опису завдань так, щоб вона була зрозуміла відповідній Грід-системі, що забезпечує роботу з різними Грід-інфраструктурами як з єдиним обчислювальним ресурсом. Така модель вимагає модифікації ППЗ, але існуючі інтерфейси можуть бути використані без істотних змін.

Розробка загального стандартного інтерфейсу для різних Грід-інфраструктур може розглядатися як універсальне рішення, яке зніме проблему функціональної сумісності, але так як кожна Грід вже сильно пов'язана зі своїм власним ППЗ, перехід на інший інтерфейс значно ускладнений. Розробка стандарту інтерфейсу для використання рядом Грід, виконується в рамках проекту European Middleware Initiative (EMI).[2]

Роботи із забезпечення функціональної сумісності і взаємодії Грід, які велися до 2010р практично у всіх країнах-учасниках Грід-проектів, не привели до створення єдиної інфраструктури, яка б дозволила використовувати обчислювальні ресурси різних національних Грід-проектів. Саме тому в 2010 р в рамках EGI був утворений окремий підпроект European Middleware Initiative (EMI) по створенню єдиного проміжного програмного програмного забезпечення Unified Middleware Distribution (UMD).

Для уніфікації ППЗ чотири консорціуми gLite, ARC, UNICORE і dCache об'єднали зусилля в спільному проекті розвитку проміжного програмного забезпечення EMI (офіційний сайт проекту http://www.eu-emi.eu/).

Зараз в Європі використовуються в промисловій експлуатації три види ППЗ: ARC, gLite, UNICORE. Вони використовуються в трьох головних грід Європи: EGI, DEISA і NDGF. Кожне із цих ППЗ виконує функціональні можливості по наданню грід-сервісів в рамках своїх проектів, але жодне з них не покриває всі потреби в Грід-обчисленнях.

Використаний термін Unified Middleware Distribution (UMD) для позначення інтегрованого дистрибутива із ППЗ gLite, ARC, UNICORE і dCache.

Використання проміжного програмного забезпечення EMI

Для використання Грід-сервісів на комп'ютерах користувачів інсталюється спеціальне програмне забезпечення - інтерфейс користувача (***User Interface – UI),*** яке забезпечує взаємодію з Грід-середовищем. Користувач взаємодіє з сервісами gLite за допомогою утиліт командного рядка. Інтерфейс користувача надає команди: для відправки завдань в Грід-систему, запит

стану завдань і кластерів, отримання даних від завершених завдань, переривання завдань та ін. Також є засоби для копіювання і видалення файлів в сховищах і каталогах реплік.

Інтерфейс користувача EMI дозволяє відправляти завдання кількома способами:

- з використанням WMS ***User Interface – UI*** за допомогою системи управління завантаженнями WMS (Workload Management System) на обчислювальні елементи CREAM CE і стандартний LCG-CE;
- за допомогою команд інтерфейсу користувача CREAM CLI безпосередньо на обраний обчислювальний елемент CREAM CE;
- за допомогою команд інтерфейсу користувача ARC CLI на обчислювальні елементи, які працюють під керуванням ППЗ ARC;
- за допомогою команд інтерфейсу користувача ARC CLI безпосередньо на вибраний обчислювальний елемент CREAM CE.

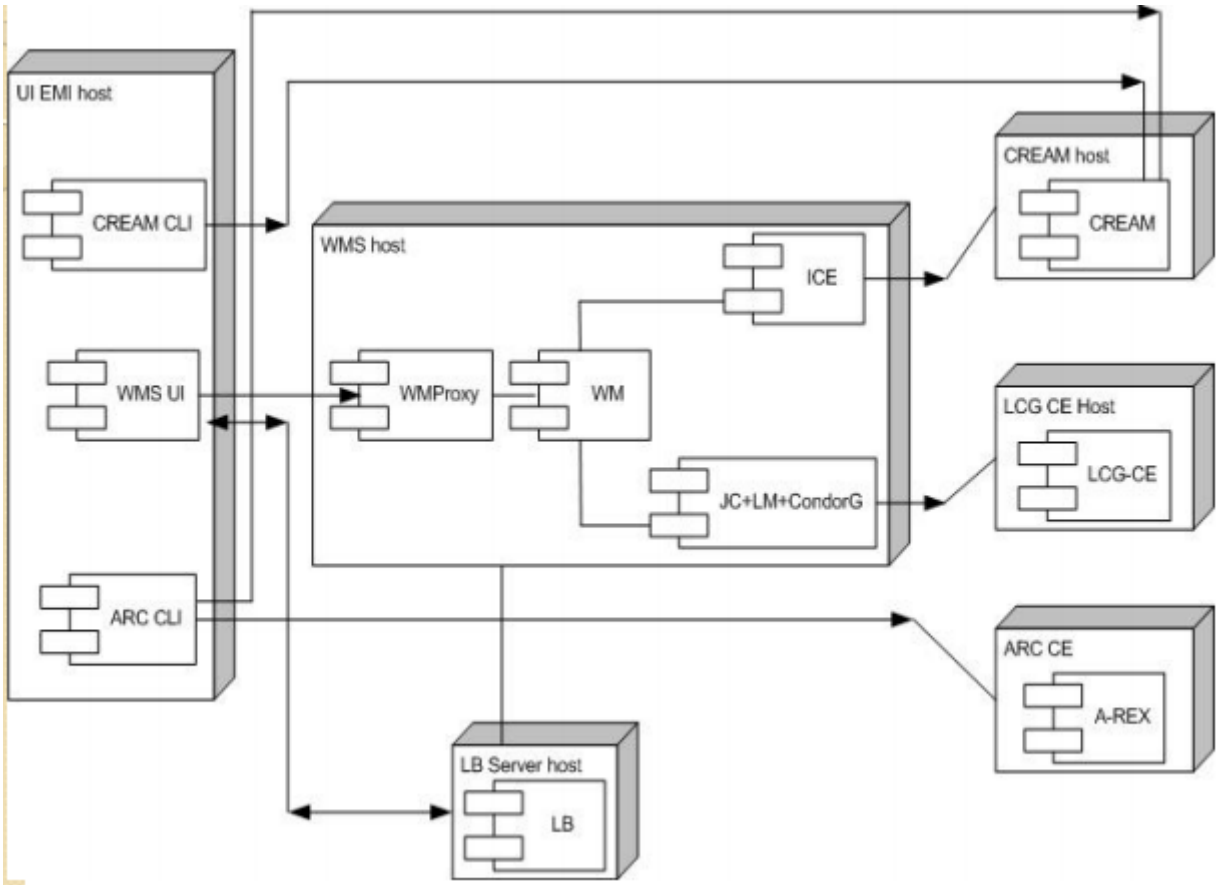


Рис.6.13 Загальна схема відправлення завдання в Грід

На рис.6.13 приведена загальна схема відправлення завдання в Грід з різними обчислювальними елементами CREAM CE, LCG-CE і ARC CE з одного інтерфейсу користувача.

Розгянемо стисло архітектуру ***Computing Resource Execution and Management (***CREAM), який включено в дистрибутив EMI-2. Більше інформації можна знайти на офіційному сайті.[2]

CREAM розроблений з використанням технології Web Services мовою програмування Java і працює як розширення Java-Axsis сервлету всередині серверу додатків Apache Tomcat.

CREAM і CEMonitor є частиною дистрибутива ППЗ gLite і використовується Грід-інфраструктурою EGI. (EGI – European Grid Infrastructure - проек, який прийшов на зміну EGEE) Користувач може встановити користувацький інтерфейс CREAM і відправляти завдання безпосередньо на обчислювальний елемент CREAM CE командами інтерфейсу користувача.

Крім того, gLite-користувач може відправляти завдання на обчислювальний елемент CREAM CE, використовуючи підсистему розподілу навантаження gLite Workload Management System (WMS). В цьому випадку має бути встановленим спеціальний компонент, названий Interface to Cream Environment (ICE). ICE приймає запити від gLite WMS і направвляє їх на CREAM CE. ICE обробляє стани завдання і реєструє їх в gLite Logging and Bookkeeping (LB) сервісі.[3]

Інтерфейс користувача CREAM UI містить набір команд, які дозволяють відправити завдання на виконання, зняти завдання, отримати звіт про поточний стан виконання завдання і скопіювати результати виконання на комп'ютер користувача.

Використання WMS дозволяє розподіляти завдання між обчислювальними елементами в Грід. При відправленні завдання безпосередньо на CREAM CE контроль стану завдання виконується спеціальним модулем ICE. ICE приймає команди від компонентів WM-системи за допомогою простого механізму повідомлень, заснованого на локальних файлах. ICE може отримувати інформацію про стан завдання двома способами. Перший спосіб використовує спеціальний компонент CEMonitor. CREAM інформує CEMonitor про кожну зміну стану завдання і потім ця інформація передається в ICE. Крім того, ICE може прямо запитувати сервіс CREAM про стан активного завдання, якщо інформація про нього не була отримана певний час. Цей механізм гарантує те, що ICE отримає інформацію про стан завдання навіть у випадку, якщо з модулем CEMonitor виникнуть проблеми.

Для забезпечення сумісності з різними Грід-системами реалізований додатковий сервіс CREAM service, заснований на Basic Execution Service (BES) специфікації. BES визначає основні операції по керуванню завданнями на обчислювальному елементі Грід і аналогічний механізм повинен бути реалізований в інших системах.

A-REX - ARC Compute Element (ARC CE)

Дистрибутив ППЗ EMI включає всі основні компоненти ARC і повністтю забезпечує функціональність з виконання завдань у Грід.

ARC Compute Element являє собою набір сервісів , які реалізують функцію доступу в грід для авторизованих користувачів, відправку завдання на виконання безпосередньо в локальну систему керування, обробку запитів користувачів і публікацію інформації про хід виконання завдання і роботу сервісу в інформаційній системі. Цей сервіс збудований на основі Grid Manager з дадаванням Web Service-інтерфейсу. Він заснований на відкритих стандартах

сумісності (включаючи інструментарій OGF Basic Execution Service –BES) і специфікації Job Submission Description Language (JDSL) і підтримує LRMSs (Local Resource Management Systems). Взаємодія інтерфейсу користувача з A-REX забезпечується компонентом ARC Core – набір базових пакетів, які забезпечують реалізацію основних сервісів ARC. Основним пакетом є Hosting Environment Deamon (HED), який являє собою бібліотеку базових сервісів, необхідних для роботи пакетів ARC. HED забезпечує базову функціональність (комунікаційні канали, доступ до бібліотек, авторизацію, аутентифікацію, парсинг конфігураційних файлів для Грід-сервісів), що дозволяє виділити загальні функції Грід-сервісів із загальної логіки і розробляти нові Грід-сервіси дуже єфективно. Інформаційна система ARC Iformation System, яка являє собою набір сервісів для публікації, зберігання і візуалізації інформації, містить в собі ARIS (ARC Resource Information System)- – сервіс публікації інформації про A-REX і про завдання, які виконуються на ресурсі в LDAP-форматі, EGIIS (Enhanced Grid Information Indexing Service) – сервіс реєстрації ARISes і LDAP. Сервіси ARIS відповідають за опис і характеристики ресурсів (ресурсів обчислень, ресурсів накопичених даних). Інформація про локальні сервіси генерується на самому ресурсі (також можливе її кешування). Доступ до цієї інформації можливий через LDAP-інтерфейс. Основним завданням сервісів EGIIS є зберігання динамічного списку ресурсів (їх LDAP-адреса). Індексні сервіси таким чином поєднуються між собою і реалізують конкретну топологію Грід-сегменту. Перед використанням кожної з команд інтерфейсу користувача EMI необхідно мати (Грід-користувачам) дійсний проксі-сертифікат, доступний системі керування завданнями, який можна створити командами ***grid-proxy-init, voms- proxy-init, arcproxy.*** Вибір команди залежить від типу викристов ППЗ. Якщи проксі-сертифікат вже є на комп'ютері зі встановленим користувацьким інтерфейсом, можна використати змінну середовища з X509_USER_PROXY з зазначенням шляху до нього. Крім цього, в каталозі *SHOME/.globus* треба мати пару ключів, тобто файли ***usercert.pem, userkey.pem*** з відповідними правами доступу.

WMS-інтерфейс користувача

Основні команди інтерфейсу WMS користувача системи gLite дистрибутиву EMI-2. (більш детальну інф. див. http://glite.web.cern.ch/glite/documenataion/)

Основні команди інтерфейсу WMS користувача

Команда	Опис команди
1	3

glite-wms-job-submit<jdl_file>	Відправка завдання на виконання
glite-wms-job-delegate-proxy < delegation_Id>	Делегування проксі-сертифікату до WMProxy та присвоєння відповідного ідентифікатора
glite-wms-job-status<job_Id>	Відображення стану виконання завдання
glite-wms-job-list-match<jdl_file>	Формування переліку обчислювальних ресурсів, які задовольняють вимогам виконання завдання і на яких авторизовано даного користувача
glite-wms-job-cancel<job_Id>	Відміна виконання завдання
glite-wms-job-output<job_Id>	Копіювання вихідних файлів після завершення завдання

Інтерфейс користувача CREAM CE EMI-2

Основні команди інтерфейсу користувача системи ***CREAM CE EMI*** . (більш детальну інф. див. http://glite.web.cern.ch/glite/documenataion/)

Основні команди *CREAM CE EMI*

Команда	Опис команди
1	3
glite-ce-job-submit<jdl_file>	Відправка завдання на виконання, опис якого міститься у файлі <jdl_file>
glite-ce-job-delegate-proxy < delegation_Id>	Делегування проксі-сертифікату до WMProxy та присвоєння відповідного ідентифікатора
glite-ce-job-status<job_Id>	Відображення стану виконання завдання
glite-ce-job-list <host[:port] >	Відображення переліку ідентифікаторів завдань, відправлених на виконання на CREAM CE користувачем
glite-ce-job-cancel<job_Id>	Відміна виконання завдання
glite-ce-job-suspend<job_Id>	Тимчасове припинення виконання раніше відправлених завдань користувача на CREAM CEs (кількість завдань має бути >=1)
glite-ce-job-resum<job_Id>	Відновлення виконання раніше відправлених завдань користувача на CREAM CEs (кількість завдань має бути >=1)
glite-ce-job-output<job_Id>	Копыювання вихыдних файлыв пысля завершення завдання

glite-ce-job-purge<job_Id>	Видалення інформації по N завданням користувача з CREAM CEs (кількість завдань має бути >=1). Після цієї операції інформація (вхідні та вихідні файли буде недоступна.
glite-ce-proxy-renew < delegation_Id>	Продовження дії проксі-сертифікату для відпралених на виконання завдань на CREAM CEs за ідентифікатором, який був створений по команді glite-ce-job-delegate-proxy
glite-ce-service-info <host[:port] >	Відображення інформації по сервісам CREAM (версія, статус та інше)
glite-ce-get-cemon-url <host[:port] >	Відображення інформації по сервісе CEMon CREAM CE
glite-ce-enable-submission <host[:port] >	Відновлення можливості відправки завдань на окремий CREAM CE
glite-ce-disable-submission <host[:port]>	Заборона на відправку завдань на окремий CREAM CE
glite-ce-allowed-submission<host[:port]>	Перевірка можливості відправки завдання на окремий CREAM CE

Перед використанням кожної з команд CREAM CE необхідно мати дійсний проксі-сертифікат, доступний системі управління завданнями. Його можна створити за допомогою команди ***voms-proxy-init.***

Опис стану завдання в CREAM

Стан завдання	Опис стану завдання
1	3
Registered	Завдання відправлено на виконання на обчислювальний елемент CREAM
pending	Завдання оброблюється
idle	Завдання знаходиться в локальній черзі LRMS та очікує відправки на виконання
running	Завдання оброблюється в LRMS
Really-running	Завдання виконується
held	виконання завдання призупинено командою користувача
Done-ok	виконання завдання завершилося успішно
Done-failed	виконання завдання завершилося з помилками
cancelltd	виконання завдання відмінено командою

	користувача
aborted	Помилка при відправленні завдання в локальну систему керування LRMS

Інтерфейс користувача ARC – EMI

Наведемо основні команди користувацького інтерфейсу ***ARC – EMI***. (більш детальну інф. див. http://www.nordugrid.org/ в розділі Документація)

Команда	Опис команди
1	3
arcproxy [options]	Генерування проксі-сертифікату
arclcs [options]	Генерування проксі-сертифікату IdP для SAML2SSO профілю
arcsub [options] [filename …]	Відправлення завдання на грід-ресурси для виконання. Завдання має бути підготовлене в одній із мов опису JSDL, xRSL, JDL
arctat [options] [job …]	Перегляд стану кластерів і завдань
arccat [options] [job …]	Перегляд потокового виведення результатів та повідомлень про помилки під час виконання завдання
arcget [options] [job …]	Копіювання вихідних файлів після завершення завдання
arcsync [options]	Синхронізація інформації про виконання завдань, яка зберігається на одному комп'ютері з інформацією на ARC clientкомп'ютері, з якого започаткована поточна сесія роботи
arcinfo [options]	Отримання статичної інформації про обчислювальний ресурс в Грід
arckill [options] [job …]	Відміна виконання завдання
arcclean [options] [job …]	Видалення файлів виведення результатів виконання завдання з кластеру, в якому вони виконувалися
arcrenew [options] [job …]	Відновлення проксі-сертифікату завдання
arcresume [options] [job …]	Відновлення роботи завдання після збою при виконанні
arcresub [options] [job …]	Повторна відправка завдання на виконання
arcmigrate [options] [job …]	відправка завдання на виконання в інший обчислювальний елемент з припиненням поточного виконання на ресурсі (використовується тільки для web-сервісу A-REX)
arcls [options] <URL>	Перегляд змісту та деяких атрибутів файлів на ресурсі зберігання даних в певній

	директорії, яка задається за допомогою URL
arccp [options] <source> <destination>	Команда копіювання файлів між грід-кластерами
arcrm [options] <URL>	Видалення файлів з ресурсу зберігання даних
arcmkdir [options] <URL>	Команда створення нової директорії на ресурсі зберігання даних, який задано за допомогою URL
arcsrmping [options] <service>	Тестування доступності SRM-сервісу (подібна команді ping в Unix)
arctest [options]	Тестування функціональності грід-ресурсу

Перед використанням кожної з команд користувацького інтерфейсу ARC – EMI потрібно мати дійсний проксі-сертифікат (створюється за допомогою команди ***arcproxy. Arcproxy*** дозволяє генерувати такі проксі-сертифікати, які використовуються в різних грід: pre-RFC GSI proxy; RFC-compliant proxy (default); VOMS-extended proxy; MyProxy delegation.

При використанні грід-ресурсів, які вимагають членства користувача в Віртуальній організації, кманда ***arcproxy*** може генерувати проксі-сертифікат з розширенням VOMS, яке засвідчує належність користувача до певної ВО та надає право доступу до її Грід-ресурсів:

```
$ arcproxy --voms atlas
Your identity: /DC=***/O=***/CN=***
Enter pass phrase for /home/user/.cert/userkey.pem:
.....++++++
.....++++++
Contacting VOMS server (named atlas): voms.cern.ch on port: 15001
Proxy generation succeeded
Your proxy is valid until: 2018-05-24 03:08:35
```

При цьому потрібно додати опис віртуальної організації в файлі ~ /***.voms/vomses.***

Опис завдання в ARC –EMI виконується мовою xRSL (Extended Rtsource Specification Language) і має містити всю необхідну для виконання на віддаленому ресурсі інформацію (файл виконання, параметри та ін.) і набір вимог, яким маэ відповідати кластер для виконання завдання (об’єм диску, встановлене програмне забезпечення) і куди мають бути записані результати виконання.

Приклад завдання:

```
&
(executable=task.sh)
(inputFiles=(task.sh "")( task.c ""))
(stdout="out.txt")
(stderr="err.txt")
(outputFiles=("out.txt" "")("err.txt" ""))
(jobname="test1")
```

Тут вказано, що в якості завдання використовується файл скрипту ***task.sh.*** На віддалений ресурс потрібно скопіювати файл ***task.sh*** і ***task.c***, а результати виконання будуть збережені в файлі ***out.txt,*** файл реєстрації помилок - ***err.txt.***

Якщо завдання успішно відправлене на виконання на віддалений обчислювальний ресурс, то йому (завданню) присвоюється ідентифікатор ***job ID,*** який відображається в термінальному вікні (standard output).

В ARC – EMI опис завдання можна виконувати мовою ***JSDL*** та ***JDL.*** Так, в прикладі ***myjob.jsdl*** наведено вивід "***Hello World***" з використанням мови ***JSDL.***

```
<version="1.0" encoding ="UTF-8"?>
<JobDefinition
xmlns="http://schemas.ggf.org/jsdl/2015/11/jsdl"
xmlns:posix="http://schemas.ggf.org/jsdl/2015/11/jsdl- posix">
<JobDescription>
        <JobIdentification>
                <JobName>Hello World job</JobName>
      </JobIdentification>
      <Application>
      <posix:POSIXApplication>
              <posix:Executable>bin/echo</posix:Executable>
              <posix:Argument>'Hello World'</posix:Argument>
              <posix:Output>out.txt</posix:Output>
              <posix:Error>err.txt</posix:Error>
       </posix:POSIXApplication>
     </Application>
</JobDescription>
</JobDefinition>
```

В наступному прикладі - опис завдання з виведенням "***Hello World***" виконано мовою JDL.

[
Executable = "script-jdl.sh";
StdOutput ="std.out";
StdError ="std.err";
OutputStandbox = {"std.out","std.err"};
OutputStandboxDestURL = {"gsiftp://localhost/std.out","gsiftp://
localhost/std.err"};
]

Файл скрипту *script-jdl.sh* має вигляд:

#!/bin/sh
Echo "Example forJDL file"

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення Грід.
2, Перерахуйте особливості програмування Грід.
3. Що таке координація розрізнених ресурсів?
4. На що націлені Грід - обчислення ?
5. Назвати обчислювальні ресурси Грід.
6. Що таке Сумісніять midlleware Грід.
7. Призначення Middleware.
8. Порівняйте суперкомп'ютери та грід-системи.
9. Назвіть відміни між архітектурами КК та Грід-систем.
10. В чому полягає ідея батьківських і дочірніх процесів?
11. Особливості програмування Грід-систем та комп'ютерних кластерів.
12. Охарактеризуйте особливості алгоритмів планування розміщення завдання в Grid і комп'ютерних кластерах.
13. Планування завдань в Грід-системах та комп'ютерних кластерах.
14 Перерахуйте найбільш відомі пакетіи проміжного програмного забезпечення Грід.

Література

1. CHPC Software: Moab Scheduler . Режим доступу: www.chpc.utah.edu/docs/manuals/software/moab.html
2. Towards Open Grid Services Architecture. Режим доступу: http://toolkit.globus.org/ogsa/
3. IBM Redbooks. Режим доступу: www.redbooks.ibm.com
4. Chee B.J.S., Franklin C.Jr. Cloud Computing. Technologies and Strategies of the Ubiquitous Data Center. CRC Press: Taylor & Francis Group. – 2010. –263 p.

5. За редакцією Загороднього А.Г. та Згуровського М.З., На шляху до європейського грід: довідник (проект). - К.: НТУУ «КПІ», 2012. − 391 с

Література допоміжна

2. Introduction to GRID Computing, December 2005. – IBM Redbook, − 241 с.
2 www.globus.org
3 Support for Open Source Globus Toolkit Ends January 2018 | globus [Electronic resource] – Режим доступу: https://www.globus.org/blog/support-open-source-globus-toolkit-ends-january-2018 – Accessed: 04.05.2019 – Назва з екрану.
4 GT 5.0.0 Release Notes [Electronic resource] – Режим доступу: http://toolkit.globus.org/toolkit/docs/5.0/5.0.0/rn/#rn-changesummaries – Accessed: 04.05.2019 – Назва з екрану.
5 Globus Toolkit 6.0 Release Manuals [Electronic resource] – Режим доступу: http://toolkit.globus.org/toolkit/docs/6.0/ – Accessed: 04.05.2019 – Назва з екрану.
6 http://lcg.web.cern.ch/LCG/
7 www.glite.org
8 www.unicore.org
9 "LCG – Deployment". Lcg.web.cern.ch. Retrieved July 29, 2010.
10 gLite - Lightweight Middleware for Grid Computing [Electronic resource] – Режим доступу: http://grid-deployment.web.cern.ch/grid-deployment/glite-web/ – Accessed: 11.05.2019 – Назва з екрану.
11 What's EMI ? - European Middleware Initiative [Electronic resource] – 2016. – Режим доступу: https://web.archive.org/web/20161015155746/http://www.eu-emi.eu/what-is-emi – Accessed: 11.05.2019 – Назва з екрану.
12.http://www.eu-emi.eu/
13.http://www.eu-emi-2-matterhorn

7 Інформаційні технології Cloud computing в високопродуктивних обчисленнях

7. 1. Cloud computing.

Вперше пішла мова про ідею хмарних обчислень на конференції LinuxWorld 2003. Тоді звернули увагу на систему Scyld, яка була призначена для реалізації сайтів та бізнес-застосувань. Її особливістю була та, що будь-який потік запитів до сайту завжди обслуговувався за рахунок того, що призупинялись малопріоритетні процеси, а важливі процеси динамічно переміщувались на ресурси КК, що вивільняються. Відміною від ОС паралельних систем була та, що система Scyld могла виконувати балансування

навантаження великої кількості різних СЕ, які мають власні ресурси та ОС різного типу.

Як і багато нових технологій, хмарні обчислення розвивались поетапно. На першому етапі хмарні обчислення розвивались як нова послуга – Software as a Service (SaaS), тобто матзабезпечення як послуга в рамках Інтернет - стандарту Web 2.0. Почало з'являтись багато хостинг-сайтів, які були призначені для зберігання сайтів, веб-застосунків осіб та компаній. З'явились системи керування контентом (content management systems, CMS) і відповідно, традиційні сайти почали масово замінюватись на сайти на базі CMS. Тобто, тут CMS є типовим матзабезпеченням як послуга на віддаленому сервері. Крім того, швидке зростання вартості електроенергії змушує багато фірм та організацій мінімізувати витрати, в тому числі використовуючи віддалений доступ до сервера.

Хмарні обчислення - це бізнес-модель, в якій обчислювальні ресурси, такі як обчислення та накопичення, упаковані як сервіси, які схожі на фізичні комунальні послуги.

Хмарні обчислення представляють собою динамічно масштабуємий спосіб доступу до зовнішних обчислювальних ресурсів в вигляді сервісу, який надається посередництвом Інтернету, при цьому користовачу не потрібно ніяких особливих знань про інфраструктуру «хмари» чи навиків управління цією «хмарною» технологією, або Cloud computing – це програмно-апаратне забезпечення, доступне через Інтернет в вигляді сервісу, позволяючого використовувати інтерфейс для віддаленого доступу до виділенх ресурсів (обчислювальних ресурсів , програм і даних). Комп'ютер користовача виступає при цьому рядовим терміналом, підключеним до мережі.

Появу хмарних обчислень обумовили дві ключові тенденції – консолідація і віртуалізація IT-інфраструктури

Консолідація – це об'єднання обчислювальних ресурсів або структур управління в єдиному центрі.

Саме вона здатна суттєво зменшити витрати на IT.

Зазвичай кажуть про консолідацію:

- **серверів** – переміщення децентралізованих додатків, розподілених на різних серверах компанії, в один кластер централізованих гомогенних серверів;
- **систем зберігання** – спільне використання централізованої системи зберігання даних декількома гетерогенними вузлами;
- **додатків** – розміщення декількох додатків на одному хості.

При цьому можна виділити два базових типи консолідації - фізичну і логічну. Фізична консолідація має на увазі географічне переміщення серверів на єдину площадку (в центр даних), а логічна - централізацію управління.

Переміщення комп'ютерів в єдиний центр обробки даних дозволяє забезпечити комфортні умови для обладнання та технічного персоналу, а також збільшити ступінь фізичного захисту серверів. Крім того, в центрі обробки

даних можна використовувати більш продуктивне і високоякісне обладнання, яке економічно недоцільно встановлювати в рядових підрозділах.

Очевидна перевага цього рішення в тому, що спрощується робота з розгортання та керування системами, знижується ступінь дублювання. Спрощується забезпечення фізичного захисту та поліпшується мережева безпека, оскільки сервери опиняються під захистом єдиного, централізовано керованого центру.

Логічний тип консолідації має на увазі перебудову системи управління IT-інфраструктури. Це необхідно як для збільшення масштабованості і керованості складної розподіленої обчислювальної системи, так і для об'єднання сегментів корпоративної мережі. Логічна консолідація додатків призводить до централізації управління критичними для бізнесу системами і додатками. Тут переваги очевидні: в першу чергу це вивільнення апаратних ресурсів, які можна використовувати на інших ділянках інформаційної системи. По-друге, більш проста і логічна структура управління IT-інфраструктурою робить її більш гнучкою і пристосованою для майбутніх змін.

Сценарій гомогенної консолідації передбачає перенесення одного масштабованого додатку, який раніше виконувався на декількох серверах, на один, більш потужний. Об'єднання даних і додатків на одному сервері помітно прискорює процеси обробки і пошуку, а також підвищує рівень цілісності.

Гетерогенна консолідація за змістом схожа з гомогенною, але в цьому випадку об'єднанню підлягають різні додатки. Наприклад, кілька примірників Exchange Server і SQL Server, раніше запускалися на окремих комп'ютерах, можуть бути зведені на єдиній машині. Переваги гетерогенної консолідації – зростаюча масштабованість сервісів і більш повне задіяння системних ресурсів.

Як відзначають фахівці з хмарних технологій - консолідація IT-інфраструктури стала першим кроком до "хмари".

Віртуалізація – це маскування від користувача складності апаратної та програмної реалізації системи та її складових, географічних відстаней між вузлами, належності вузлів різним організаціям, створення ілюзії роботи з реальним суперкомп’ютером. По статистиці средній рівень завантаження процессорных потужностей у серверів під управлінням Windows не превершує 10%, у Unix-систем цей показник кращий, але і там в средньому не превершує 20%. Низька ефективність використання серверів пояснюється широким використанням з початку 90-х років підходом “один додаток — один сервер”, т. ч. щоразу для розгортання нового додатку використовували новий сервер. Очевидно, що на практиці це призводить до швидкого збільшення серверного парку і як наслідок — до зростанню затрат на його адміністрування, енергоспоживання та охолодження, а такж потреби в збільшенні приміщень для установки нових серверів та отриманні ліцензій на серверну ОС.

В основі віртуалізації лежить можливість одного комп'ютера виконувати роботу декількох комп'ютерів завдяки розподілу його ресурсів на декілька середовищ. За допомогою віртуальних серверів і віртуальних настільних комп'ютерів можна розмістити кілька ОС і кілька додатків в єдиному розташування. Таким чином, фізичні та географічні обмеження перестають мати якесь значення.

У комп'ютерних технологіях під терміном «віртуалізація» зазвичай розуміється абстракція обчислювальних ресурсів і надання користувачеві системи, яка «інкапсулює» (приховує в собі) власну реалізацію. Простіше кажучи, користувач працює зі зручним для себе представленням об'єкту і для нього не має значення, як об'єкт влаштований в дійсності. Можливість запуску декількох віртуальних машин на одній фізичній викликає інтерес серед комп'ютерних фахівців не тільки тому, що це підвищує гнучкість IT – інфраструктури, але й тому, що це дає значну економічну вигоду.

Переваги технологій віртуалізації

1. Ефективне використання обчислювальних ресурсів.
2. Скорочення витрат на інфраструктуру. Віртуалізація дозволяє скоротити кількість серверів і пов'язаного з ними IT -обладнання в інформаційному центрі.
3. Зниження витрат на програмне забезпечення. Деякі виробники програмного забезпечення ввели окремі схеми ліцензування спеціально для віртуальних середовищ.
4. Підвищення гнучкості і швидкості реагування системи. Віртуалізація пропонує новий метод управління IT – інфраструктурою і допомагає IT – адміністраторам витрачати менше часу на виконання повторюваних завдань – наприклад, на ініціацію, налаштування, відстеження і технічне обслуговування.
5. **Несумісні додатки можуть працювати на одному комп'ютері**. При використанні віртуалізації на одному сервері можлива установка Linux і Windows серверів, шлюзів, баз даних і інших абсолютно несумісних в рамках однієї невіртуалізованої системи додатків.
6. Підвищення доступності додатків і забезпечення безперервності роботи підприємства. Завдяки надійній системі резервного копіювання та міграції віртуальних середовищ цілком без перерв в обслуговуванні Ви зможете скоротити періоди планового простою і забезпечити швидке відновлення системи в критичних ситуаціях. "Падіння» одного віртуального сервера не веде до втрати інших віртуальних серверів.
7. Можливості легкої архівації. Оскільки жорсткий диск віртуальної машини зазвичай представляється у вигляді файлу певного формату, розташованого на якому-небудь фізичному носії, віртуалізація дає можливість простого копіювання цього файлу на резервний носій як засіб архівування і резервного копіювання всієї віртуальної машини.
8. Підвищення керованості інфраструктури: використання централізованого управління віртуальною інфраструктурою дозволяє скоротити час на адміністрування серверів, забезпечує балансування навантаження і "живу" міграцію віртуальних машин.

Віртуальна машина – це повністю ізольований програмний контейнер, який працює з власною ОС і додатками, подібно фізичному комп'ютеру.

Віртуальна машина діє так само, як фізичний комп'ютер, і містить власні віртуальні (тобто програмні) оперативну пам’ять, жорсткий диск і мережевий

адаптер. ОС не може розрізнити віртуальну і фізичну машини. Те ж саме можна сказати про додатки та інших комп'ютерах в мережі. Але віртуальні машини складаються виключно з програмних компонентів і не включають обладнання. Це дає їм низку унікальних переваг.

Основні різновиди віртуалізації

Розглянемо основні різновиди віртуалізації, такі як:
- віртуалізація серверів (повна віртуалізація і паравіртуалізація)
- віртуалізація на рівні операційних систем
- віртуалізація додатків,
- віртуалізація уявлень.

Віртуалізація серверів

Віртуалізація серверів має на увазі запуск на одному фізичному сервері декількох віртуальних серверів. Віртуальні машини або сервери являють собою програми, запущені на хостовій операційній системі, які емулюють фізичні пристрої серверу. На кожній віртуальній машині може бути встановлена операційна система, на яку можуть бути встановлені додатки і служби. Типові представники - це продукти VmWare (ESX, Server, Workstation) і Microsoft (Hyper -V, Virtual Serer, Virtual PC).

Повна віртуалізація (Full, Native Virtualization). Використовуються немодифіковані екземпляри гостьових операційних систем, а для підтримки роботи цих ОС служить загальний шар емуляції їх виконання поверх хостової ОС, в ролі якої виступає звичайна операційна система. Така технологія застосовується, зокрема, в VMware Workstation, VMware Server (колишній GSX Server), Parallels Desktop, Parallels Server, MS Virtual PC, MS Virtual Server, Virtual Iron. До переваг даного підходу можна вважати відносну простоту реалізації, універсальність і надійність рішення; всі функції управління бере на себе хост-ОС. Недоліки – високі додаткові накладні витрати на використовувані апаратні ресурси, відсутність обліку особливостей гостьових ОС, менша, ніж потрібно гнучкість у використанні апаратних засобів.

Паравіртуалізації (paravirtualization). Модифікація ядра гостьової ОС виконується таким чином, що в неї включається новий набір API, через який вона може безпосередньо працювати з апаратурою, не конфліктуючи з іншими віртуальними машинами. При цьому немає необхідності задіяти повноцінну ОС в якості хостового ПЗ, функції якого в даному випадку виконує спеціальна система, що отримала назву гіпервізора (hypervisor). Саме цей варіант є сьогодні найбільш актуальним напрямком розвитку серверних технологій віртуалізації і застосовується в VMware ESX Server, Xen (і рішеннях інших постачальників на базі цієї технології), Microsoft Hyper -V. Переваги даної технології полягають у відсутності потреби в хостової ОС – ВМ встановлюються фактично на "голе залізо".

Віртуалізація на рівні ядра ОС (operating system – level virtualization). Цей варіант передбачає використання одного ядра хостової ОС для створення незалежних паралельно працюючих операційних середовищ. Для гостьового ПЗ створюється тільки власне мережеве та апаратне оточення. Такий варіант використовується в Virtuozzo (для Linux і Windows), OpenVZ (безкоштовний варіант Virtuozzo) і Solaris Containers. Переваги – висока ефективність використання апаратних ресурсів, низькі накладні технічні витрати, відмінна керованість, мінімізація витрат на придбання ліцензій. Недоліки – реалізація тільки однорідних обчислювальних середовищ.

Віртуалізація додатків має на увазі застосування моделі сильної ізоляції прикладних програм з керованою взаємодією з ОС, при якій віртуалізується кожен екземпляр додатків, всі його основні компоненти: файли (включаючи системні), реєстр, шрифти, INI – файли, COM – об'єкти, служби. Додаток виповнюється без процедури інсталяції в традиційному її розумінні і може запускатися прямо з зовнішніх носіїв (наприклад, з флеш – карт або з мережевих папок). З точки зору IT – відділу, такий підхід має очевидні переваги: прискорення розгортання настільних систем і можливість управління ними, зведення до мінімуму не тільки конфліктів між додатками, а й потреби у тестуванні додатків на сумісність. Дана технологія дозволяє використовувати на одному комп'ютері, а точніше в одній і тій же операційній системі кілька несумісних між собою додатків одночасно.

Віртуалізація уявлень (робочих місць). Віртуалізація уявлень має на увазі емуляцію інтерфейсу користувача. Тобто користувач бачить додаток і працює з ним на своєму терміналі, хоча насправді додаток виконується на віддаленому сервері, а користувачеві передається лише зображення віддаленої програми. Залежно від режиму роботи користувач може побачити віддалений робочий стіл і запущений на ньому додаток, або тільки саме вікно програми.

Слід вказати, що історія розвитку технологій віртуалізації налічує 50 років. Компанія IBM була першою, хто задумався про створення віртуальних середовищ для різних користувальницьких завдань (тоді ще в мейнфреймах). У 60-х роках минулого століття віртуалізація представляла чисто науковий інтерес і була оригінальним рішенням для ізоляції комп'ютерних систем в рамках одного фізичного комп'ютера. Після появи персональних комп'ютерів інтерес до віртуалізації дещо послабився незважаючи бурхливий розвиток операційних систем, які пред'являли адекватні вимоги до апаратного забезпечення того часу. Однак бурхливе зростання апаратних потужностей комп'ютерів наприкінці дев'яностих років минулого століття змусив IT-спільноту знову згадати про технології віртуалізації програмних платформ.

У 1999 р компанія VMware представила технологію віртуалізації систем на базі x86 в якості ефективного засобу, здатного перетворити системи на базі x86 в єдину апаратну інфраструктуру загального користування та призначення, що забезпечує повну ізоляцію, мобільність і широкий вибір ОС для прикладних середовищ. Компанія VMware була однією з перших, хто зробив серйозну ставку виключно на віртуалізацію. Як показав час, це виявилося абсолютно виправданим. Сьогодні WMware пропонує комплексну віртуалізаційну платформу четвертого покоління VMware vSphere 4, яка включає ресурси як

для окремого ПК, так і для центру обробки даних. Ключовим компонентом цього програмного комплексу є гіпервізор VMware ESXServer. Пізніше в "битву" за місце в цьому модному напрямку розвитку інформаційних технологій включилися такі компанії як Parallels (раніше SWsoft), Oracle (Sun Microsystems), Citrix Systems (XenSourse).

Корпорація Microsoft вийшла на ринок засобів віртуалізації в 2003 р з придбанням компанії Connectix, випустивши свій перший продукт Virtual PC для настільних ПК. З тих пір вона послідовно нарощувала спектр пропозицій у цій галузі і на сьогодні майже завершила формування віртуалізаційних платформ , до складу якої входять такі рішення як Windows 2008Server R2 з компонентом Hyper-V, Server 2012 Hyper-V, Microsoft Application Virtualization (App-v), Microsoft Virtual Desktop Infrastructure (VDI),Remote Desktop Services, System Center Virtual MachineManager .

На сьогоднішній день постачальники технологій віртуалізації пропонують надійні і легкокеровані платформи, а ринок цих технологій переживає справжній бум. За оцінками провідних експертів, зараз віртуалізація входить до трійки найбільш перспективних комп'ютерних технологій.

Короткий огляд платформ віртуалізації.

VMware

Компанія VMware - одна з перших гравців на ринку платформ віртуалізації. В1998 році VMware запатентувала свої програмні техніки віртуалізації і з тих пір випустила чимало ефективних і професійних продуктів для віртуалізації різного рівня: від VMware Workstation, призначеного для настільних ПК, доVMware ESX Server, що дозволяє консолідувати фізичні сервери підприємства в віртуальній інфраструктурі. За результатами різних тестів продуктивності засоби віртуалізації VMware майже завжди за більшістю параметрів виграють у конкурентів. VMware має більше 100 000 клієнтів по всьому світу. Мережа партнерств охоплює понад 350 виробників обладнання та програмного забезпечення і понад 6000 реселерів.

Платформи віртуалізації VMware:

VMware Workstation - платформа, орієнтована на desktop-користувачів і призначена для використання розробниками ПЗ, а також професіоналами в сфері IT. Нова версія популярного продукту VMware Workstation 7 стала доступна в 2009 р, в якості хостових операційних систем підтримуються Windows і Linux. VMware Workstation 7 може використовуватися спільно з середовищем розробки, що робить її особливо популярною в середовищі розробників, користувачів і фахівців технічної підтримки. Вихід VMware Workstation 7 означав офіційну підтримку Windows 7 як в якості гостьової, так і хостової операційної системи. Продукт включає підтримку Aero Peek і Flip 3D, що робить можливим спостерігати за роботою віртуальної машини, наближаючи курсор до панелі завдань VMware, або до відповідної вкладки на робочому столі хоста. Версія може працювати на будь-якій версії Windows 7, а також як і будь-які версії Windows можуть бути запущені в віртуальних машинах. Крім того, віртуальні машини в VMware Workstation 7 повністю

підтримують Windows Display Driver Model (WDDM), що дозволяє використовувати інтерфейс Windows Aero в гостьових машинах.

VMware Player - безкоштовний «програвач» віртуальних машин на основі віртуальної машини VMware Workstation, призначений для запуску вже готових зразків віртуальних машин, створених в інших продуктах VMware, а також в Microsoft VirtualPC і Symantec LiveState Recovery. Починаючи з версії 3.0 VMware Player дозволяє також створювати зразки віртуальних машин. Обмеження функціональності стосується в основному функцій, призначених для IT-фахівців і розробників ПЗ.

VMware Fusion - настільний продукт для віртуалізації на платформі Mac від компанії Apple.

VMware Server, Безкоштовний продукт VMware Server є досить потужною платформою віртуалізації, яка може бути запущена на серверах під управлінням хостових операційних систем Windows і Linux. Основне призначення VMware Server - підтримка малих і середніх віртуальних інфраструктур невеликих підприємств. У зв'язку з невеликою складністю його освоєння і установки, VMware Server може бути розгорнутий в найкоротші терміни як на серверах організацій, так і на комп'ютерах домашніх користувачів.

VMware Ace - продукт для створення захищених політиками безпеки віртуальних машин, які потім можна поширювати по моделі SaaS (Software-as-aService).

VMware vSphere - комплекс продуктів, що представляє надійну платформу для віртуалізації ЦОД. Компанія позиціонує даний комплекс також як потужну платформу віртуалізації для створення і розгортання приватної «хмари». VMware vSphere поставляється в декількох випусках з можливостями, призначеними спеціально для малих підприємств і середніх компаній і корпорацій. VMware vSphere включає ряд компонентів, що перетворюють стандартне обладнання в загальне стійке середовище, яке нагадує мейнфрейм і включає вбудовані елементи управління рівнями обслуговування для всіх додатків:

• Служби інфраструктури - це компоненти, які забезпечують всебічну віртуалізацію ресурсів серверів, сховищ і мереж, їх об'єднання та точне виділення додатків на вимогу і відповідно до пріоритетів бізнесу.

• Служби додатків - це компоненти, що надають вбудовані елементи управління рівнями обслуговування для всіх додатків на платформі платформи vSphere незалежно від їх типу чи ОС.

• **VMware vCenter Server** надає центральну консоль для управління віртуалізацією, що забезпечує адміністрування служб інфраструктури і додатків. Ця консоль підтримує всебічну візуалізацію всіх аспектів віртуальної інфраструктури, автоматизацію повсякденної експлуатації і масштабованість для керування великими середовищами ЦОД (рис.7.1.1).

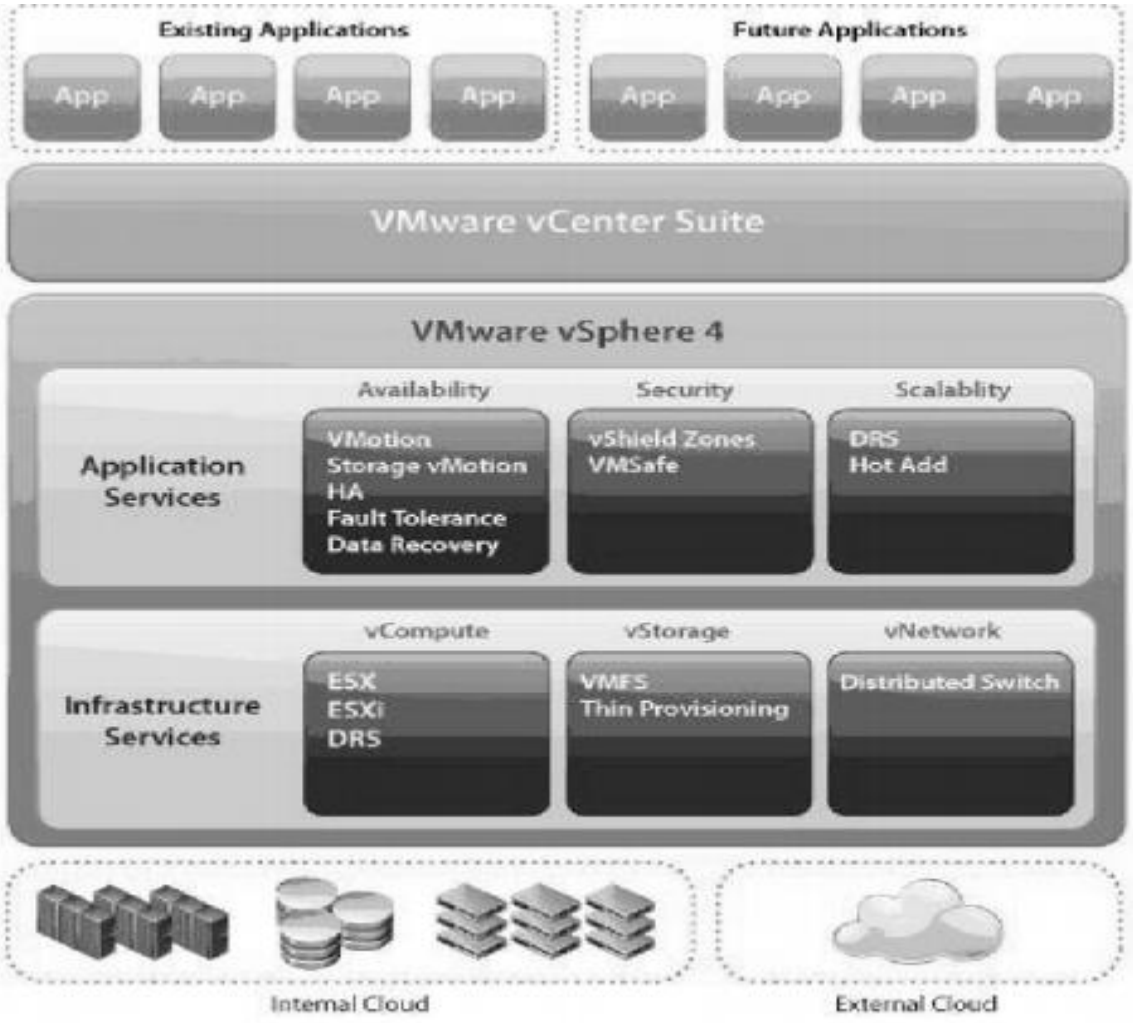


Рис.7.1.1 Структура платформи **vSphere**

VMware ESX Server - це гипервизор, який розбиває фізичні сервери на безліч віртуальних машин. VMware ESX є основою пакету VMware vSphere і входить в усі випуски VMware vSphere (рис.7.1.2)

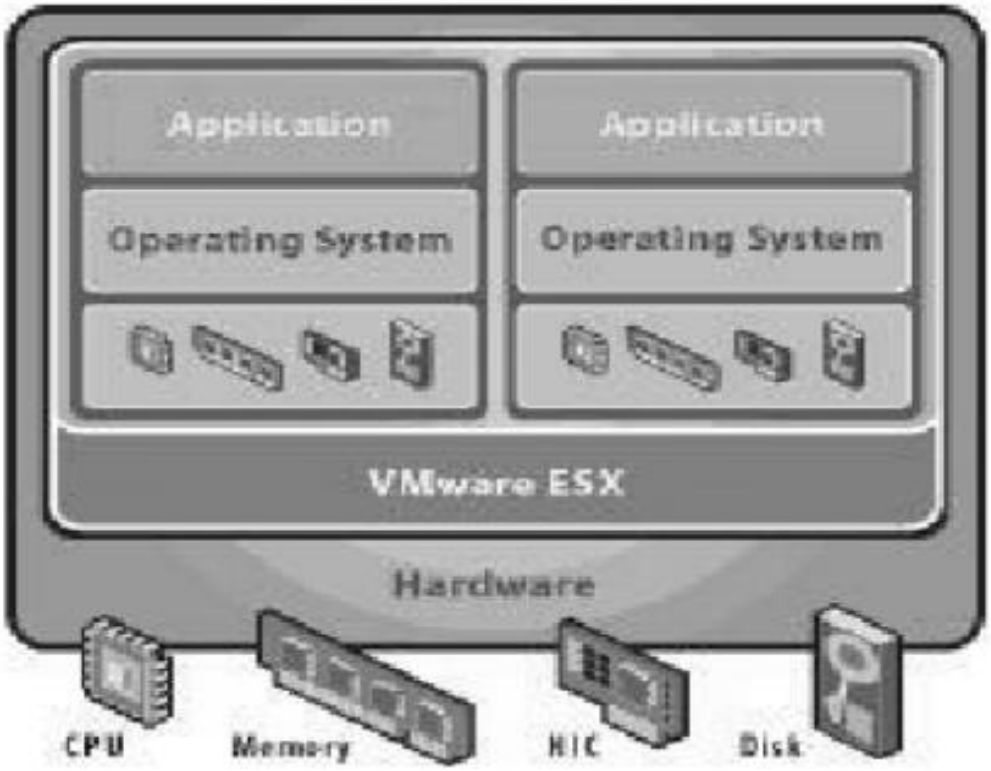


Рис.7.1.2 Гіпервізор VMware ESX

VMware vSphere Hypervisor (раніше VMware ESXi) - "полегшена" платформа віртуалізації корпоративного рівня, заснована на технологіях ESX. Продукт є безкоштовним і доступний для завантаження з сайту VMware. VSphere VMwareHypervisor є найпростішим способом для початку роботи з віртуалізацією.

VMware vCenter- надає розширюєму і масштабовану платформу для опереджувального управління віртуальною інфраструктурою і забезпечує отримання про неї всеосяжної інформації. VMware vCenter Server забезпечує централізоване управління середовищами vSphere і спрощує виконання повсякденних завдань, значно покращуючи адміністративне управління середовищем. Продукт володіє широкими можливостями по консолідації серверів, їх налаштування і управління. VMware vCenter Server агрегує в собі всі аспекти управління віртуальним середовищем: від віртуальних машин до збору інформації про фізичні сервери для подальшої їх міграції в віртуальну інфраструктуру. Крім центрального продукту управління віртуальною інфраструктурою vCenter Server існує також ряд доповнень, що реалізують різні аспекти планування, управління та інтеграції розподіленої віртуальної інфраструктури (VMware vCenter Server Heartbeat, VMware vCenter Orchestrator, VMware vCenter Capacity IQ, VMware vCenter Site Recovery Manager, VMware vCenter Lab Manager, VMware vCenter Configuration Manager, VMware vCenter Converter). Зокрема, vCenter Converter призначений для перекладу в віртуальне середовище фізичних серверів, що дозволяє здійснювати «гарячу» (без зупину систем) і «холодну» міграцію.

vCenter Site Recovery Manager - це ПЗ для створення територіально-віддаленого резервного сегменту віртуальної інфраструктури, який в разі відмови основного вузла, бере на себе функції по запуску віртуальних машин відповідно до плану відновлення після збоїв. **vCenter Lab Manager** - продукт для створення інфраструктури зберігання і доставки конфігурацій віртуальних машин, що дозволяє організувати ефективну схему тестування в компаніях-розробників ПЗ.

VMware ThinApp - колишній продукт Thinstall Virtualization Suite, ПЗ для віртуалізації додатків, що дозволяє поширювати встановлені додатки на клієнтські робочі станції, скорочуючи час на стандартні операції по встановленню і конфігурації.

VMware View - комплекс продуктів, який забезпечує централізацію призначених для користувача робочих станцій в віртуальних машинах на платформі vSphere. Це дозволяє скоротити витрати на стандартні IT-операції, пов'язані з розгортанням і обслуговуванням призначених для користувача десктопів.

VMware Capacity Planner - засіб централізованого збору та аналізу даних про апаратне і програмне забезпечення серверів, а також продуктивності обладнання. Ці дані використовуються авторизованими партнерами VMware для побудови планів консолідації віртуальних машин на платформі VMware ESX Server.

VMware VMmark - продукт, доступний тільки виробникам апаратного забезпечення, призначений для тестування продуктивності VMware ESX Server на серверних платформах.

Citrix (Xen)

Розробка некомерційного гипервизора Xen починалася як дослідницький проект комп'ютерної лабораторії Кембриджського університету. Засновником проекту і його лідером був Ian Пратт (Ian Pratt) співробітник університету, який створив згодом компанію XenSource, яка займається розробкою комерційних платформ віртуалізації на основі гіпервізора Xen, а також підтримкою Open Source співтовариства некомерційного продукту Xen. Спочатку Xen був найрозвинутішою платформою, яка підтримує технологію паравіртуалізації. Безкоштовна версія Xen включена в дистрибутиви декількох ОС, таких як Red Hat, Novell SUSE, Debian, Fedora Core, Sun Solaris. В середині серпня 2007 року компанія XenSource була поглинена компанією Citrix Systems. Сума проведеної операції близько 500 мільйонів доларів говорить про серйозні наміри Citrix щодо віртуалізації.

Безкоштовний Xen. В даний час Open Source версія платформи Xen застосовується в основному в освітніх і дослідницьких цілях. Деякі вдалі ідеї, реалізовані численними розробниками з усього світу, находять своє відображення в комерційних версіях продуктів віртуалізації компанії Citrix. Зараз безкоштовні версії Xen включаються в дистрибутиви багатьох Linux-систем, що дозволяє їх користувачам застосовувати віртуальні машини для ізоляції програмного забезпечення в гостьових ОС з метою його тестування і вивчення проблем безпеки, без необхідності установки платформи віртуалізації. До того ж багато незалежних розробники ПЗ можуть поширювати його за допомогою віртуальних шаблонів, в яких вже встановлена і налаштована гостьова система і пропонований продукт. Крім того, Xen ідеально підходить для підтримки старого програмного забезпечення у віртуальній машині. Для більш серйозних цілей у виробничому середовищі необхідно використовувати комерційні платформи компанії Citrix.

Citrix XenApp - призначений для віртуалізації і публікації додатків з метою оптимізації інфраструктури доставки сервісів у великих компаніях. XenApp має величезну кількість користувачів по всьому світу і в багатьох компаніях являється ключовим компонентом IT-інфраструктури.

Citrix XenServer - платформа для консолідації серверів підприємств середнього масштабу, яка включає основні можливості для підтримки віртуальної інфраструктури. Виробник позиціонує даний продукт як рішення Enterprise-рівня для віртуалізації серверів, яке підтримує роботу в «хмарному» оточенні.

Citrix XenDesktop - рішення по віртуалізації десктопів підприємства, яке дозволяє централізовано зберігати і доставляти робочі оточення в віртуальних машинах користувачам. Продукт підтримує кілька сценаріїв доставки додатків на настільні ПК, тонкі клієнти і мобільні ПК і сумісний з серверними віртуалізаційними рішеннями конкурентів.

Microsoft

Для Microsoft все почалося, коли в 2003 році вона придбала компанію Connectix, одну з небагатьох компаній, виробляючу програмне забезпечення для віртуалізації під Windows. Разом з Connectix, компанії Microsoft дістався продукт Virtual PC, який конкурував тоді з розробками компанії VMware щодо настільних систем віртуалізації. За великим рахунком, Virtual PC надавав тоді таку кількість функцій, як і VMware Workstation, і при належній увазі міг би бути в даний час повноцінним конкурентом цієї платформи. Однак з того часу компанія Microsoft випускала по мінорному релізу в рік, не приділяючи особливої уваги продукту Virtual PC, в той час як VMware стрімко розвивала свою систему віртуалізації, перетворивши її по-справжньому в професійний інструмент. Усвідомивши своє технологічне відставання в сфері віртуалізації серверних платформ, компанія Microsoft випустила продукт Virtual Server 2005, націлений на створення і консолідацію віртуальних серверів організацій. Однак було вже пізно. Компанія VMware вже захопила лідерство в цьому сегменті ринку, пропонуючи в той момент дві серверних платформи віртуалізації VMware GSX Server і VMware ESX Server, кожна з яких за багатьма параметрами перевершувала платформу Microsoft. Остаточний удар був нанесений в 2006 році, коли VMware фактично оголосила продукт VMware GSX Server безкоштовним, взявшись за розробку продукту VMware Server на його основі і сконцентрувавши всі зусилля на продажах потужної корпоративної платформи VMware ESX Server в складі віртуальної інфраструктури Virtual Infrastructure 3. У компанії Microsoft був тільки єдиний вихід у цій ситуації: в квітні 2006 року вона також оголосила про безкоштовність продукту Microsoft Virtual Server 2005. Також існуючі раніше два видання Standard Edition і Enterprise Edition були об'єднані в одне - Microsoft Virtual Server Enterprise Edition. З тих пір Microsoft істотно змінила стратегію щодо віртуалізації і влітку 2008 року був випущений фінальний реліз платформи віртуалізації Microsoft Hyper-V, інтегрованої в ОС Windows Server 2008. Тепер роль сервера віртуалізації доступна всім користувачам нової серверної операційної системи Microsoft. (свою гіпервізорну технологію Microsoft назвала Hyper-V)

Microsoft Virtual Server. Серверна платформа віртуалізації Microsoft Virtual Server може використовуватися на сервері під керуванням операційної системи Windows Server 2003 і призначена для одночасного запуску декількох віртуальних машин на одному фізичному хості. Платформа безкоштовна і надає тільки базові функції.

Microsoft Virtual PC. Продукт Virtual PC був куплений корпорацією Microsoft разом з компанією Connectix і вперше під маркою Microsoft був випущений як Microsoft Virtual PC 2004. Купуючи Virtual PC і компанію Connectix, компанія Microsoft будувала далекосяжні плани щодо забезпечення користувачів інструментом для полегшення міграції на наступну версію операційної системи Windows. Тепер Virtual PC 2007 безкоштовний і доступний для підтримки настільних ОС у віртуальних машинах.

Microsoft Hyper-V. Продукт Microsoft позиціонується як основний конкурент VMware ESX Server в області корпоративних платформ віртуалізації.

Microsoft HyperV є рішення для віртуалізації серверів на базі процесорів з архітектурою x64 в корпоративних середовищах. На відміну від продуктів Microsoft Virtual Server або Virtual PC, Hyper-V забезпечує віртуалізацію на апаратному рівні, з використанням технологій віртуалізації, вбудованих в процесори. Hyper-V забезпечує високу продуктивність, практично рівну продуктивності однієї операційної системи, яка працює на виділеному сервері. Hyper-V поширюється двома способами: як частина Windows Server 2008, або в складі незалежного безкоштовного продукту Microsoft Hyper-V Server.

У Windows Server 2008 технологія Hyper-V може бути розгорнута як в повній установці, так і в режимі Server Core. Hyper-V Server працює тільки в режимі Core. Це дозволяє в повній мірі реалізувати всі переваги «тонкої», економічної і керованої платформи віртуалізації.

Hyper-V є вбудованим компонентом 64-розрядних версій Windows Server 2008 Standard, Windows Server 2008 Enterprise і Windows Server 2008 Datacenter. Ця технологія недоступна в 32-розрядних версіях Windows Server 2008, в Windows Server 2008 Standard без Hyper-V, Windows Server 2008 Enterprise без Hyper-V, Windows Server 2008 Datacenter без Hyper-V, в Windows Web Server 2008 і Windows Server 2008 для систем на базі Itanium.

Розглянемо коротко особливості архітектури Hyper-v. Hyper-v є гіпервізор, тобто прошарок між обладнанням і віртуальними машинами рівнем нижче операційної системи. Ця архітектура була спочатку розроблена IBM в 1960-і роки для мейнфреймів і порівняно недавно стала доступною на платформах x86 / x64, як частина ряду рішень, включаючи Windows Server 2008 Hyper-V, Windows Server 2012 Hyper-V і Vmware ESX.

Віртуалізація на базі гіпервізора заснована на тому, що між обладнанням і віртуальними машинами з'являється прошарок, що перехоплює звернення операційних систем до процесора, пам'яті та інших пристроїв. При цьому доступ до периферійних пристроїв в різних реалізаціях гіпервізора може бути організований по різному. З точки зору існуючих рішень для реалізації менеджера віртуальних машин можна виділити два основних види архітектури гіпервізора: **мікроядерну і монолітну.**

Монолітний підхід розміщує гіпервізор в єдиному рівні, який також включає більшість необхідних компонентів, таких як ядро, драйвери пристроїв і стек введення / виведення. Це підхід використовується такими рішеннями, як VMware ESX і традиційними системами мейнфреймів.

Монолітний підхід має на увазі, що всі драйвери пристроїв поміщені в гіпервізор. У монолітної моделі гіпервізор для доступу до обладнання використовує власні драйвери. Гостьові ОС працюють на віртуальних машинах поверх гіпервізора. Коли гостьовій системі потрібен доступ до обладнання, вона повинна пройти через гіпервізор і його модель драйверів. Зазвичай одна з гостьових ОС грає роль адміністратора або консолі, в якій запускаються компоненти для надання ресурсів, управління і моніторингу всіх гостьових ОС, які працюють на сервері.

Модель монолітного гіпервізора забезпечує прекрасну продуктивність,але має ряд недоліків, таких як:

• Стійкість - якщо в оновлену версію драйвера вкралася помилка, то збої почнуться у всій системі, у всіх її віртуальних машинах.
• Проблеми оновлення драйверів - при необхідності поновлення драйверу будь-якого пристрою (наприклад мережевого адаптеру) оновити драйвер можливо тільки разом з виходом нової версії гіпервізору, в яку буде інтегрований новий драйвер для даного пристрою.
• Труднощі з використанням непідтримуваного обладнання. Наприклад, Ви зібралися використовувати обладнання «Сервер» досить потужний і надійний, але при цьому в гіпервізорі не виявилося потрібного драйвера для RAID-контролера або мережевого адаптера. Це унеможливить використання відповідного обладнання, а, значить, і серверу.

В мікроядерній реалізації можна говорити про «тонкий гіпервізор», в цьому випадку в ньому зовсім немає драйверів. Замість цього драйвери працюють в кожному індивідуальному розділі, щоб будь-яка гостьова ОС мала можливість отримати через гіпервизор доступ до обладнання. При цьому кожна віртуальна машина займає абсолютно окремий розділ, що позитивно позначається на захищеності і надійності. У мікроядерній моделі гіпервізора (в віртуалізації Windows Server 2008 R2 використовується саме вона) один розділ є кореневим, або батьківським (Parent), решта - дочірніми (child). Розділ - це найменша ізольована одиниця, підтримувана гіпервізором. Розмір гіпервизора Hyper-V менше 1,5 Мб.

Кожному розділу призначаються конкретні апаратні ресурси – частка процесорного часу, обсяг пам'яті, пристрої та ін. Батьківський розділ створює дочірні розділи і керує ними, а також містить стек віртуалізації (virtualization stack), який використовується для управління дочірніми розділами. Батьківський розділу створюється першим і володіє всіма ресурсами, які не належать Гіпервізору. Володіння всіма апаратними ресурсами означає, що саме кореневий (тобто, батьківський) розділ керує живленням, підключенням самоналагоджувальних пристроїв, відає питаннями апаратних збоїв і навіть керує завантаженням гіпервізора (рис.7.1.3).

У коренеому розділі міститься стек віртуалізації - набір програмних компонентів, розташованих поверх гипервизора, які спільно з ним забезпечують роботу віртуальних машин. Стек віртуалізації обмінюється даними з гіпервізором і виконує всі функції по віртуалізації, які не підтримуються безпосередньо гіпервізором. Велика частина цих функцій пов'язана зі створенням дочірніх розділів і управлінням ними і необхідними їм ресурсами (ЦП, пам'ять, пристрої).

Перевага мікроядерного підходу, застосованого в Windows Server 2008 R2, в порівнянні з монолітним підходом полягає в тому, що драйвери, які повинні розташовуватися між кореневим розділом і фізичним сервером, не вимагають внесення жодних змін до модель драйверів. Іншими словами, в системі можна просто застосовувати існуючі драйвери. У Microsoft цей підхід обрали, оскільки необхідність розробки нових драйверів сильно загальмувала б розвиток системи. Що ж стосується гостьових ОС, вони будуть працювати з емуляторами або синтетичними пристроями.

З іншого боку, мікроядерна модель може дещо програвати монолітній моделі в продуктивності. Однак в наші дні головним пріоритетом стала безпеку, тому для більшості компаній цілком прийнятна втрата кількох відсотків в продуктивності заради скорочення фронту нападу і підвищення стійкості. .

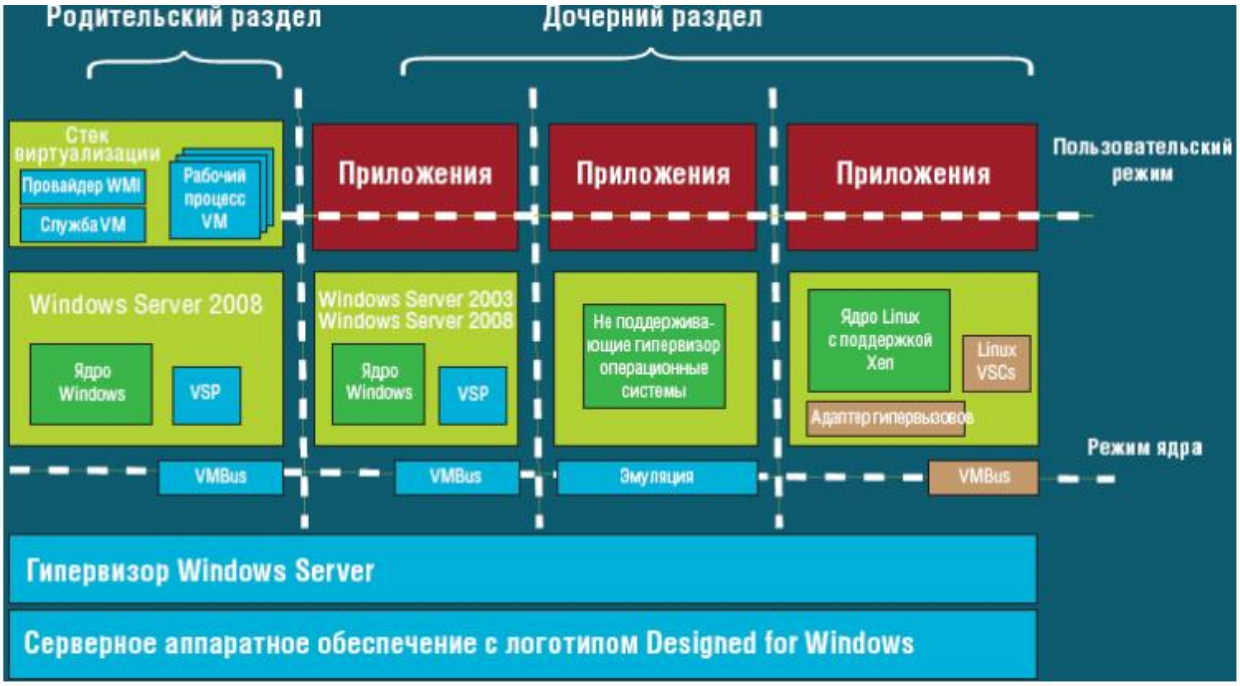


Рис.7.1.3. *Архітектура Hyper-v*

Усі версії Hyper-V мають один кореневий розділ. Цей розділ керує функціями Hyper-V. З батьківського розділу запускається консоль Windows Server Virtualization. Крім того, батьківський розділ використовується для запуску віртуальних машин (VM), що підтримують потокову емуляцію старих апаратних засобів. Такі VM, побудовані на готових шаблонах, емулюючих апаратні засоби, є аналогами VM, працюючими в продуктах з віртуалізацією на базі хоста, наприклад Virtual Server.

Гостьові VM запускаються з дочірніх розділів Hyper-V. Дочірні розділи підтримують два типи VM: високопродуктивні VM на основі архітектури VMBus і VM, керовані системою-хостом. До першої групи входять VM з системами Windows Server 2003, Windows Vista, Server 2008 і Linux (підтримують Xen).

Нову архітектуру VMBus відрізняє високопродуктивний конвеєр, функціонуючий в оперативній пам'яті, з'єднуючий клієнтів Virtualization Service Clients (VSC) на гостьових VM з провайдером Virtual Service Provider (VSP) хоста. VM, керовані хостом, запускають платформи, які не підтримують нову архітектуру VMBus: Windows NT, Windows 2000 і Linux (без підтримки технології Xen, наприклад SUSE Linux Server Enterprise 10).

Microsoft System Center Virtual Machine Manager (SCVMM) – окремий продукт сімейства System Center для управління віртуальною інфраструктурою, ефективного використанням ресурсів фізичних вузлів, а також спрощення підготовки та створення нових гостьових систем для адміністраторів і

користувачів. Продукт забезпечує всебічну підтримку консолідації фізичних серверів в віртуальній інфраструктурі, швидке і надійне перетворення фізичних машин в віртуальні, розумне розміщення віртуальних навантажень на підходящих фізичними вузлах, а також єдину консоль для управління ресурсами і їх оптимізації.

SCVMM забезпечує наступні можливості:

• Централізоване управління серверами віртуальних машин в масштабах організації. SCVMM підтримує управління серверами Microsoft Hyper-V, Microsoft Virtual Server, VMware ESX.

• Створення бібліотеки шаблонів віртуальних машин. Шаблони віртуальних машин є наборами образів встановлених операційних систем, які можуть бути розгорнуті за лічені хвилини.

• Моніторинг та розміщення віртуальних машин у відповідність із завантаженістю фізичних серверів.

• Міграція (конвертація) фізичних серверів у віртуальні машини -технологія P2V. Технологія P2V дозволяє зробити перенесення фізичного сервера на віртуальний без зупинки роботи. Таким чином, з'являється можливість онлайнового резервування цілого сервера, і в разі виходу його з ладу, можна протягом хвилини запустити віртуальний сервер і продовжити роботу.

• Міграція (конвертація) віртуальних машин інших форматів в віртуальні машини Hyper-V - технологія V2V. Дана технологія аналогічна P2V, але при цьому дозволяє переносити віртуальні машини Microsoft Virtual Server або VMware ESX в Hyper-V.

• Управління кластерами Hyper-V.

Cloud computing. *Огляд та визначення*

Єдиного і однозначного визначення хмарних обчислень не існує, але основним є те що концепція Cloud Computing – це такий підхід до розміщення, надання і споживання прикладних програм і комп'ютерних ресурсів, при якому прикладне програмне забезпечення і ресурси стають доступні через Інтернет у вигляді сервісів, які використовуються на різних платформах і засобах. Оплата таких сервісів здійснюється за їх фактичним використанням.

Незважаючи на те, що ця концепція не нова, сучасні технології роблять її значно більш ефективною завдяки розподілу фізичних систем за допомогою серверної віртуалізації. Cloud Computing надає користувачам доступ до потужних обчислювальних та накопичувальних ресурсів без необхідності знання того, де ці ресурси розташовані і як вони налаштовані.

Cloud Computing та Grid

Приведене визначення хмарних обчислень перетинається з багатьма існуючими технологіями, такими як Grid, Утилітарні обчислення, Сервісні обчислення та розподілені обчислення загалом. Але хмарні обчислення не тільки перетинаються з Grid-обчисленнями, вони насправді походять від Grid-обчислень та будуються на них, як на їх основі та інфраструктурній підтримці. Розвиток є результатом переміщення фокусу від інфраструктури, що надає ресурси зберігання та обчислення (як у випадку Grid), до тієї, що заснована на економії та націлена надавати більш абстрактні ресурси та сервіси (як у випадку Хмар). Хмарні обчислення не є новою парадигмою обчислюваної

інфраструктури; навпаки, це бізнес модель, в якій обчислювальні ресурси, такі як обчислення та накопичення, упаковані як сервіси, що схожі на фізичні комунальні послуги, такі як електрика та комутована телефонна мережа загального користування. Утилітарні обчислення є типовим прикладом використання іншої обчислюваної інфраструктури з додатковими сервісами калькуляції та моніторингу. Інфраструктура Хмари може використовуватися внутрішньо компанією чи надаватися суспільству як утилітарні обчислення.

Існує декілька факторів, що сприяють розвитку та росту зацікавленості у Хмарних обчисленнях:

- значне підвищення цін на апаратне забезпечення та збільшення обчислюваної потужності та накопичувального об'єму, поява багатоядерної архітектури та суперкомп'ютерів;
- експоненціальне зростання розміру даних у науковому інструментальному комплексі/моделюванні, видавництві та архівації в інтернеті;
- широко поширене використання обчислювальних сервісів та прикладних програм сучасного інтернету.

Щоб побачити, як Cloud Computing задовольняє вищеописані потреби, розглянемо його основні характеристики.

Основні характеристики хмарних обчислень

Масштабованість

Введення нових продуктів і сервісів вимагають від інформаційних систем витримувати зростаючі навантаження і обробляти великі обсяги даних. Необхідна швидка і надійна робота, що виключає відмови в обслуговуванні, затримки у відповідях від системи і збої. Масштабоване програмне забезпечення дозволяє витримувати велике навантаження, за рахунок збільшення кількості одночасно стартованих екземплярів. Як правило, для одночасного запуску безлічі екземплярів використовується типове обладнання, що знижує загальну вартість володіння і спрощує супроводження інфраструктури.

Еластичність

Гнучка реакція на мінливі умови роботи є однією з характеристик успішності проекту. Наприклад, може виникнути необхідність швидко впровадити новий продукт або послугу, провівши при цьому повний цикл планування, проектування та розробки інформаційної системи. Еластичність дозволяє швидко наростити потужність інфраструктури, без необхідності проведення початкових інвестицій в обладнання та програмне забезпечення. Еластичність пов'язана з масштабованістю програм, тому що вирішує завдання моментальної зміни кількості обчислювальних ресурсів, що виділяються для роботи інформаційної системи.

Мультітенантність

Мультітенантність - це один зі способів зниження витрат за рахунок максимального використання загальних ресурсів для обслуговування різних груп користувачів, різних організацій, різних категорій споживачів і т.п. Мультітенантность може бути особливо приваблива для розробників

прикладного програмного забезпечення, тому що дозволяє знизити власні витрати на оплату ресурсів хмарної платформи і максимально використовувати доступні обчислювальні ресурси.

Сплата за використання

Оплата використаних ресурсів - це ще один атрибут хмарних обчислень, що дозволяє оптимізувати витрати на проект, купуючи тільки необхідний обсяг ресурсів. У поєднанні з мультітенантністю, розділяючи ресурси між різними споживачами, можна знизити витрати ще більше. Еластичність дозволить швидко змінити обсяг ресурсів у бік збільшення чи зменшення, тим самим, привівши витрати на IT у відповідність з фактичними потребам організації.

Самообслуговування

Традиційно, розгортання інформаційної системи передує визначенням специфікації обладнання, його закупівлею і настроюванням. У залежності від того, ким проводиться процес розробки програми, він може вимагати виділення апаратних ресурсів і установку програмного забезпечення. Все це може зайняти тривалий час. Самообслуговування дозволяє споживачам запросити і одержати необхідні ресурси за лічені хвилини.
Отже, тільки поєднання декількох атрибутів хмарних обчислень призводить до досягнення завдання підвищення доходів та зниження витрат. Так, оплата тільки використаних ресурсів максимально ефективна в поєднанні з еластичністю інфраструктури. Еластичність, у свою чергу, припускає, що прикладне програмне забезпечення масштабується, в іншому випадку, швидке виділення ресурсів не призведе до підвищення продуктивності.
Потрібно також розуміти, що перехід в хмару часто потребує перегляду та зміни архітектури існуючих рішень, а іноді - повної відмови від них на користь створення нових, реалізованих з урахуванням можливостей, що надаються хмарними платформами. У залежності від архітектури існуючих програм та технологій, на яких вони реалізовані, їх перенесення на хмарну платформу може призвести до отримання ряду переваг, а може - до появи додаткових проблем, пов'язаних, наприклад, із забезпеченням сумісності або обмеженнями реалізації серверної платформи на рівні хмари.

Технічні рішення Cloud Computing

Слід розуміти, що під словосполученням "хмарна платформа" як правило поєднують різні, відмінні між собою, технічні рішення. Далі наведено первинну класифікацію хмарних платформ згідно з архітектурою та потребами, яким вони задовольняють.
Платформи прийнято розділяти на три групи по рівню віртуалізації:

- Хмарні хостинги
- Хмарні контейнери
- Хмарні сервіси

Відповідно до цих рівнів виділяють три моделі надання послуг: IaaS, PaaS, Saas. Детальніше розібратися у цих рівнях допоможе базова абстрактна модель архітектури «хмари».

Архітектура Cloud Computing

«Хмара» є не єдиним сервісом, а набором сервісів. Кожен рівень на рис. 7.1.4 вносить свій вклад у надання сервісу.

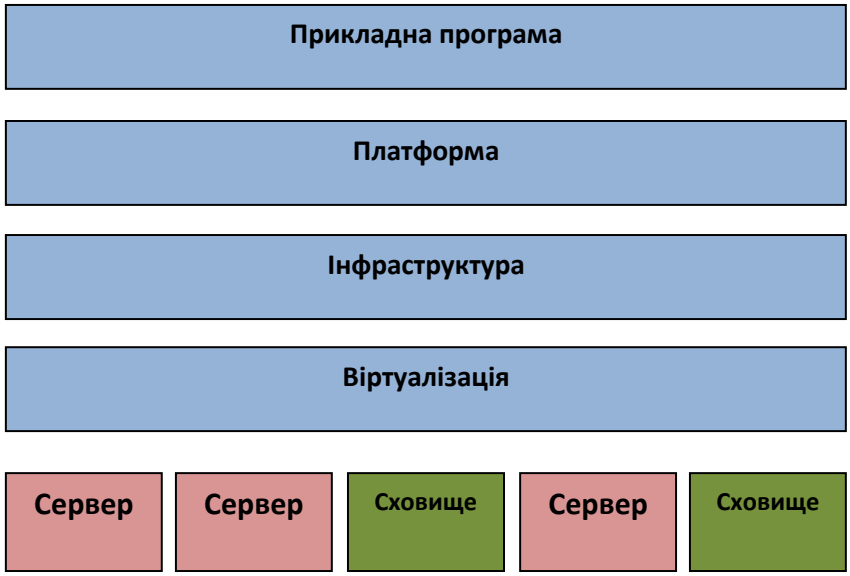


Рис. 7.1.4. Рівні, на яких будується Cloud Computing

Розглянемо найнижчий рівень, що відповідає за інфраструктуру (Infrastructure-as-a-Service або IaaS, інфраструктура як сервіс). IaaS є сервісом, який орендує інфраструктуру, тобто обчислювальні ресурси і системи зберігання. До цих ресурсів відносяться не лише віртуальні сервери з гарантованою обчислювальною потужністю, але й канали зв'язку необхідної пропускної здатності для доступу до сховищ даних та Інтернет. На цьому рівні надається можливість тимчасового використання комп'ютерів або дата-центрів при необхідній якості обслуговування, з можливістю виконання довільної операційної системи і програм.

Наступний рівень сервісу при русі по діаграмі вгору - рівень платформи (Platform-as-a-Service або PaaS, платформа як сервіс). PaaS схожий на рівень IaaS, але включає операційні системи і супутні служби, орієнтовані на певні прикладні програми. Наприклад, PaaS спільно з віртуальними серверами і системами зберігання надає певну операційну систему і набір програм, а також доступ до різних спеціалізованих локальних сервісів. Іншими словами, PaaS - це IaaS разом із стеком програм, що виконує конкретне завдання.

Зверху схеми розташовується найбільш простий рівень, що надається, - рівень прикладних програм (Software-as-a-Service або SaaS, програмне забезпечення як сервіс), який припускає використання програми з централізованої (та, можливо, віддаленої - з "хмари") системи для роботи на локальному комп'ютері. SaaS є вимірюваною послугою і дозволяє орендувати програму і оплачувати тільки час роботи з нею.

Далі буде детальніше розглянуто порівняльну характеристику платформ на різних рівнях віртуалізації, та конкретні платформи, що працюють на цих рівнях.

Хмарні хостинги, IaaS

Iaas, Infrastructure as a Service, це надання комп'ютерної інфраструктури як послуги на основі концепції хмарних обчислень.

Даний вид платформ надає можливість вирішити проблему керування ресурсами на рівні операційної системи, дозволяючи розгорнути прикладну програму практично будь-якої архітектури. Особливістю такого підходу є динамічне виділення ресурсів на основі віртуалізації операційної системи.

Ця група платформ з'явилася раніше всіх та існує відносно давно, тому такі технічні рішення мають уже налагоджені й відпрацьовані механізми.

IaaS складається з трьох основних компонентів:

1. Апаратні засоби (сервери, системи зберігання даних, клієнтські системи, мережеве обладнання)

2. Операційні системи та системне ПЗ (засоби віртуалізації, автоматизації, основні засоби управління ресурсами)

3. Сполучне ПЗ (наприклад, для управління системами)

Технології віртуалізації дозволяють взяти устаткування і розділити його на частини, які відповідають поточним потребам проекту, тим самим збільшуючи утилізацію наявних потужностей. Це зумовлює перехід від придбання, управління і амортизації апаратних активів до придбання серверного часу, дискового простору, мережевої пропускної здатності, необхідної для виконання програми. Таким чином, віртуалізація дозволяє реалізувати весь набір функцій управління в одній інтегрованій платформі.

Представники: Amazon EC2, Azure, Eucaliptus, Scalaxy, Clodo.

Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) - сервіс, який надає обчислювальні потужності у «хмарі». Сервіс входить у інфраструктуру Amazon Web Services.

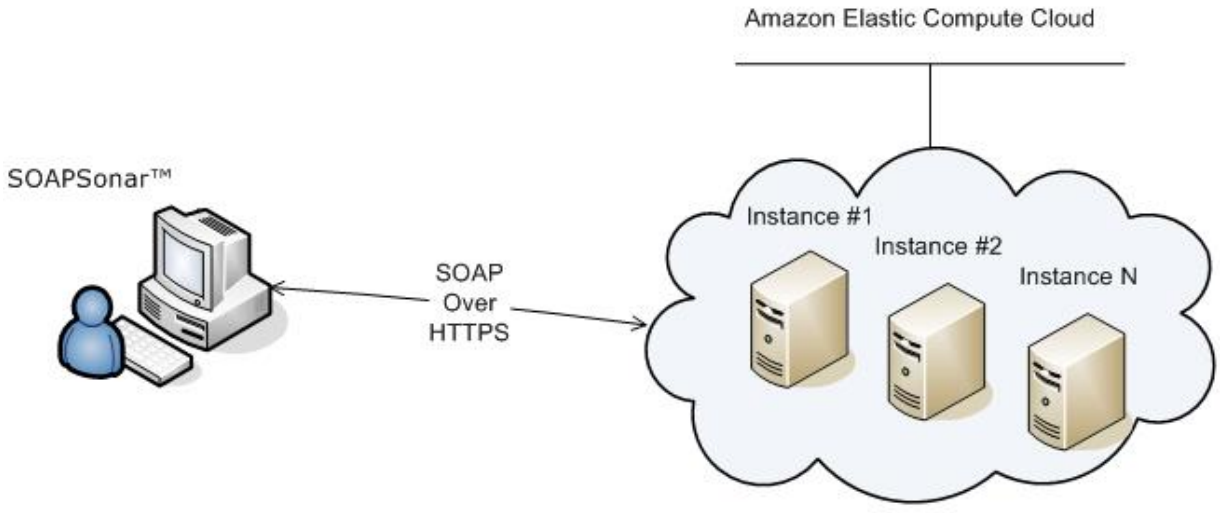


Рис. 7.1.5. Взаємодія користувача та «хмари» Amazon EC2

Простий web-інтерфейс сервісу дозволяє одержати доступ до обчислювальних потужностей і настроїти з мінімальними витратами ресурсів. Він надає користувачам повний контроль над обчислювальними ресурсами, а також доступне середовище для роботи. Сервіс суттєво скорочує час, необхідний для одержання, завантаження та налаштування нового сервера.
EC2 дозволяє:

- Створити Amazon Machine Image (AMI), який буде містити прикладні сервіси проекту, бібліотеки, дані та пов'язані з ними конфігураційні параметри, або обрати один із багатьох AMI, що пропонує Amazon;
- Завантажити AMI в Amazon S3. Amazon S3 , що забезпечує безпечне, надійне й швидке сховище для зберігання даних;
- Використовувати Amazon EC2 web-сервіс для настроювання безпеки й мережного доступу;
- Обирати необхідний тип операційної системи, запустити, завершити, або контролювати AMI у міру необхідності, використовуючи API web-сервісу, або інші інструменти керування;
- Сплачувати тільки за ресурси що споживаються, такі як процесорний час або передача даних.

Amazon Simple Storage Service (Amazon S3)

Amazon S3 - онлайнова web-служба, запропонована компанією Amazon Web Services, що надає можливість для зберігання та отримання будь-якого обсягу даних, в будь-який час з будь-якої точки мережі. За допомогою Amazon S3 досягається висока масштабованість, надійність, висока швидкість і недорога інфраструктура зберігання даних.
Вперше з'явилася в березні 2006 року в США і в листопаді 2007 року в Європі. Amazon S3 використовується багатьма сервісами для зберігання та хостингу файлів. Наприклад, це сервіси зберігання та обміну файлів Dropbox і Ubuntu One, web-сайти Twitter і Woot.com.
Amazon S3 навмисно побудована з мінімальним набором функцій, такими як:

- Записувати, читати і видаляти об'єкти, що містять від 1 байта до 5 гігабайт даних. Кількість об'єктів, які дозволено зберігати не обмежена.
- Кожен об'єкт зберігається у певному сегменті пам'яті, так званому "контейнер" і отримує від розробника унікальний ключ.
- Кожен сегмент можна зберігати в одному з кількох "регіонів". Є можливість вибрати регіон для оптимізації затримки, мінімізувати витрати, або адреси нормативним вимогам. Amazon S3 в даний час доступний у США, Ірландії та Азіатсько-Тихоокеанськогу регіоні (Сінгапур).
- Об'єкти, що зберігаються в регіоні ніколи не покидають його, поки користувач не перемістить їх.
- Механізми аутентифікації надаються для гарантії того, що дані надійно захищені від несанкціонованого доступу. Об'єкти можуть бути приватними або державними, і права можуть бути надані для певних користувачів.

- Передбачена гнучкість послуги, так що завжди можуть бути додані нові протоколи або функціональні шари. Протокол який використовується за замовчанням — HTTP.

Таким чином, Amazon пропонує на гарних умовах віртуальну машину зі статичним IP, гнучкою конфігурацією і root доступом, що дозволяє розгортання на даній платформі від легких тестових web-проектів (micro instance передбачує рік безкоштовного використання), до потужних розподілених систем (наприклад Dropbox).

Microsoft Azure

Компанія Microsoft вступила на арену Cloud Computing з платформою служб Azure, у яку входять такі служби:

- Windows Azure (операційна система як служба)
- SQL Azure (база даних хмари)
- Служби .NET
- Microsoft Live Services - набір сервісів для роботи з документами. Забезпечує зберігання, поширення і синхронізацію документів, фотографій та інших файлів між комп'ютерами, телефонами, програмами та web-сайтами;
- Microsoft SharePoint Services - набір сервісів для спільної роботи над проектами;
- Microsoft Dynamics CRM Services - сервіси для управління інформацією та взаємовідносинами з клієнтами.

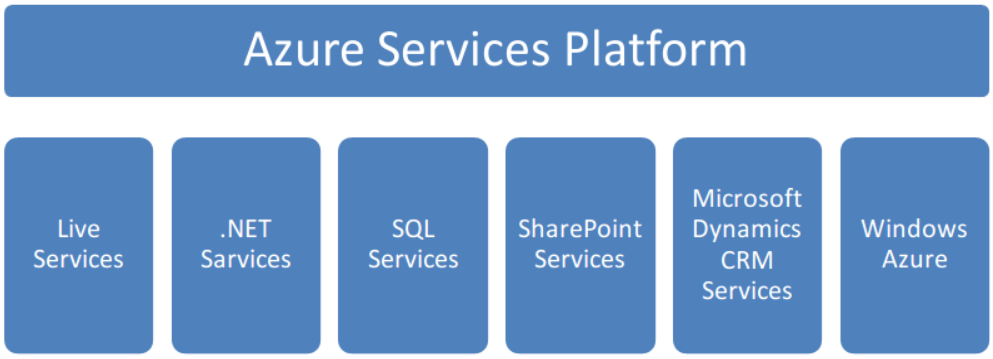


Рис. 7.1.6 Складові частини платформи Microsoft AzurePlatform

Дана платформа може використовуватися web-прикладними програмами, запущеними в хмарі (Windows Azure) або на локальному комп'ютері. Windows Azure запускається на машинах дата-центрів, і до них можна отримати доступ через Інтернет.

При розробці програми можна використовувати будь-яку . NET мову. Після написання коду сервісу, треба підготувати пакет до публікації, що складається з двох етапів:

- Завантаження проекту в Staging для перевірки.
- Переклад перевіреного проекту в Production

Виконання прикладної програми

У Windows Azure програма має кілька примірників, кожен з яких виконує частину або весь код програми. Кожен з примірників працює на своїй віртуальній машині. Але сама прикладна програма на Windows Azure не бачить

віртуальну машину, на якій вона працює. Замість цього розробник може створити web-прикладні програми з двома типами екземплярів - *Web Role* і *Worker Role*.

Примірник Web Role приймає вхідні запити через Internet Information Services (IIS). Worker Role отримує вихідні дані від Web Role, зазвичай через чергу в сховищі даних Windows Azure. Результат роботи примірників Worker Role може записуватись в сховище Windows Azure або надсилатися у зовнішній світ через вихідні мережні з'єднання.

Кожна віртуальна машина, на якій працюють екземпляри Web Role або Worker Role, містить *агента* Windows Azure, через якого web-прикладні програми взаємодіють з протоколом Fabric Windows Azure. Кожній віртуальній машині відповідає своє фізичне ядро процесора. Завдяки цьому можна керувати продуктивністю кожної прикладної програми збільшуючи кількість працюючих екземплярів вказаних в конфігураційному файлі.

Безпека в Microsoft Azure

Підхід Microsoft до безпеки в хмарному середовищі грунтується на документі Securing Microsoft's Cloud Infrastructure (Забезпечення безпеки інфраструктури хмари Microsoft).

Компанія застосовує механізми безпеки на різних рівнях хмарної інфраструктури, щоб реалізувати інформаційний захист на глибокому рівні. Ці різнорівневі механізми включають в себе:

- Фізичну безпеку дата-центрів (замки, камери, біометричні пристрої, кард-рідери, сигналізація);
- Брандмауери, шлюзи прикладних програм і IDS для захисту мережі;
- Access Control Lists (ACL - списки контролю доступу), що застосовуються до віртуальних локальних мереж (VLAN) і прикладних програм;
- Аутентифікація і авторизація людей і процесів, що запитують доступ до даних;
- Зміцнення захисту серверів та операційних систем;
- Надлишкова інфраструктура внутрішніх і зовнішніх DNS з обмеженим правом на запис;
- Забезпечення безпеки об'єктів віртуальних машин;
- Забезпечення безпеки статичних і динамічних пристроїв зберігання.

Всі ресурси компанії поділяються на категорії за рівнями необхідної інформаційної безпеки, грунтуючись на потенційних масштабах збитків. Високоцінні активи захищаються більш суворо, наприклад, за допомогою багатофакторної аутентифікації (смарт-карти, біометрія, апаратні маркери). Також реалізується принцип найменших привілеїв, коли людям і процесам надається найменший необхідний їм для виконання своєї роботи рівень доступу, і не більше того.

Хмарні контейнери, PaaS

PaaS, Platform as a Service - на відміну від IaaS, призначено більше для розробників ПЗ і дозволяє абстрагуватися від архітектури інфраструктури. У хмарі функціонує деякий набір програм, основних сервісів і бібліотек, на основі яких пропонується розробляти свої прикладні програми. Найяскравіший

приклад – платформа для створення програм Google App Engine. Крім цього, під PaaS розуміють також і окремі частини складних систем, на зразок системи бази даних чи комунікацій.

На відміну від хмарних хостингів, контейнери платформи реалізують завдання динамічного виділення ресурсів на рівні web-серверу. Вони дозволяють розгортати високомасштабовані web-сервіси прикладного призначення, обмежуючи вибір мови програмування та набір доступних бібліотек і модулів.

Дана група з'явилася небагато пізніше попередньої, і деякою мірою доповнює її. Технічна реалізація хмарних контейнерів часто будується поверх хмарних хостингів, але це приховано від користувачів на рівні контейнеру web-сервера.

Представники: GAE, Heroku.

Google App Engine

App Engine від Google - інфраструктура, яка дозволяє виконувати прикладні web-сервіси, написані на Python або Java, на серверах Google. GAE вирішує багато стандартних задач web-розробки, таких як організація сховища даних і пошуку по них; пропонується набір API для сервісів зберігання даних, акаунтів Google, завантаження даних по URL, електронної пошти і т.д. Надається можливість використовувати планувальник завдань cron. Середовище виконання включає в себе повну реалізацію можливостей самого Python, більшість функцій його стандартної бібліотеки, обмежену версію фреймворку Django і т.д.

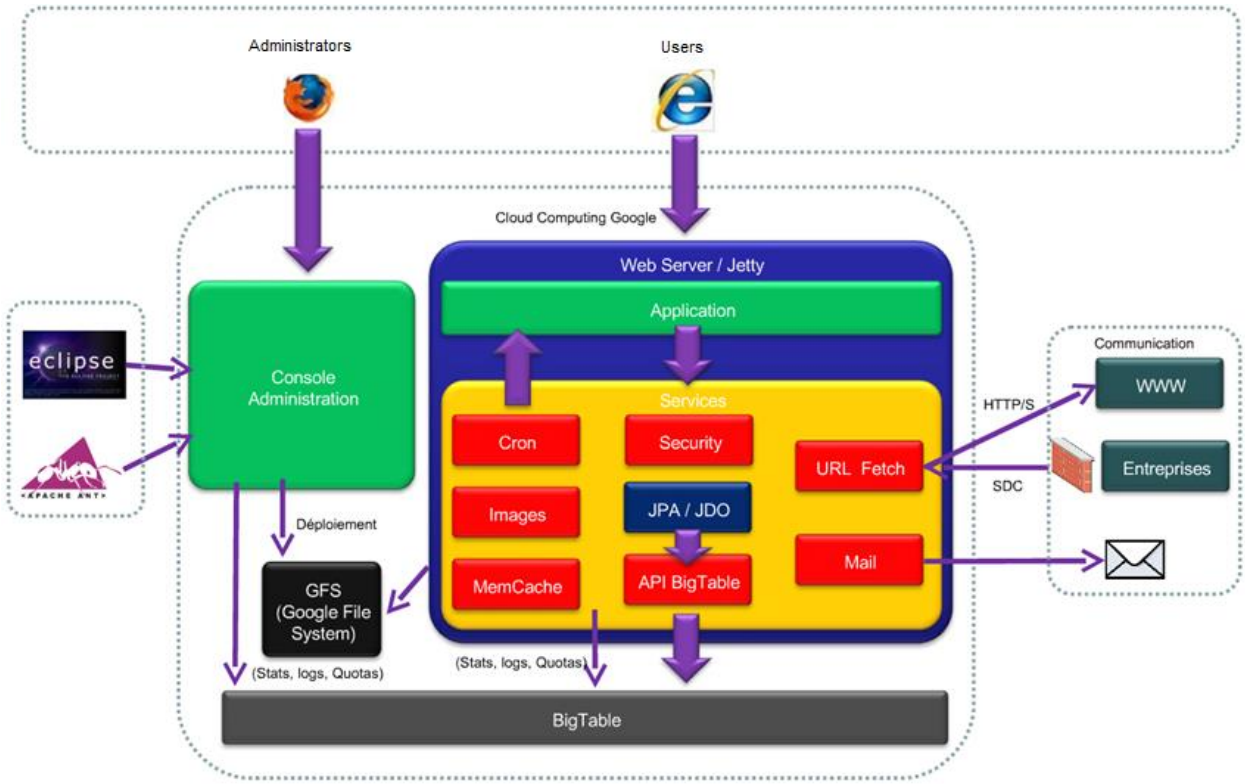


Рис. 7.1.7 Архітектура Google App Engine

На відміну від традиційних хостингів на віртуальних машинах, таких як Amazon EC2, платформа App Engine тісно інтегрована з додатками і накладає

на розробників деякі обмеження. Конкуруючі платформи хмарного хостингу дозволяють оперувати безліччю програмного забезпечення, створеного під *NIX системи, у той час як App Engine вимагає від розробника обов'язкового використання мов програмування Python або Java і збереження інформації у власному сховищі (Datastore) - підмножині пропрієтарної бази даних BigTable від Google.

Хоча більшість із розроблених на мові Python прикладного програмного забезпечення можливо запустити без змін під App Engine, використання сховища вимагає значних сил від розробників на адаптацію прикладного програмного забезпечення. На відміну від популярних СУБД MySQL і PostgreSQL, сховище не є реляційної базою даних, зокрема не підтримує схему і не вимагає нормалізації даних. Архітектура платформи підтримує масштабування без зміни коду. Реалізація підтримки реляційної логіки призвело б до значного уповільнення роботи. Нереляційний характер сховища вимагає від розробників використання нетрадиційної архітектури зберігання та обробки даних. Наприклад, замість нормалізованого зберігання даних у двох або кількох таблицях з наступним об'єднанням у запиті (join), логічно пов'язані дані рекомендується зберігати в одній таблиці.

Середовище виконання платформи Google App Engine дозволяє розробнику контролювати лише частину параметрів операційної системи. Немає доступу на запис у файлову систему сервера. Єдиний спосіб зберігати дані - внутрішнє сховище, отже, системи управління сесіями, які використовують файлову систему, не будуть працювати. Google надає альтернативний варіант - Memcache API. Є можливість організувати авторизацію користувачів через акаунти Google.

Але незважаючи на наведені обмеження, які є скоріше вимушеними кроками зі сторони Google для забезпечення надійності і високої продуктивності платформи, Google App Engine надає простий спосіб створення прикладного програмного забезпечення, здатних обслуговувати мільйони користувачів, без необхідності розширення інфраструктури та додаткових витрат для її обслуговування. Google широко використовує у своїх програмах технологію MapReduce, високошвидкісний інтерфейс для паралельних обчислень, проведених на безлічі вузлів кластера. Таким чином, погоджуючись з деякими обмеженнями, накладеними Google на програмування прикладних web-сервісів, розробник отримує потужну і високомасштабовану систему без будь-яких додаткових зусиль.

Хмарні сервіси

Крім організації ефективного споживання ресурсів, хмарні сервіси в першу чергу призначені для вирішення принципово іншого питання, а саме для усунення проблеми довготривалості, технічних і економічних складностей при розробці. Хмарні сервіси зазвичай реалізуються поверх хмарних контейнерів чи хмарних хостингів, і в своїй основі містять принцип віртуалізації прикладного програмного забезпечення всередині платформи.

Розробка прикладного програмного забезпечення на базі хмарних сервісів максимально орієнтована на використання API готового набору сервісів.

Представники: Google Apps, Oracle Service Bus, Longjump, Saleforce Service Cloud.

Google Apps

Google Apps - служби, що надаються компанією Google для використання свого доменного імені з деякими продуктами Google.

Рис. 7.1. 8 Служби Google

Служба підтримує декілька web-прикладних програм зі схожою функціональністю як у традиційних офісних пакетів, і включає:

1. Gmail;

Базовий пакет безкоштовний і пропонує той же об'єм електронної скриньки для одного користувача, що і звичайний обліковий запис Gmail. Має розширену технічну підтримку, надає API для інтеграції з існуючою інфраструктурою і гарантію 99,9% часу безперебійної роботи електронної пошти.

2. Google Calendar;

Управління порядком денним, складання розкладів, спільне використання інтернет-календарів, налаштування доступу, синхронізація з мобільними пристроями.

3. Google Talk (чат);
4. Google Docs;

Документи, електронні таблиці та презентації: робота в Інтернеті без прикріплених файлів.

5. Google Groups

Користувальницькі групи з поштовими розсилками, простим доступом до інформації та можливістю пошуку по архівах.

6. Google Sites.

Створення web-сторінок, сайтів та баз знань без навичок програмування.

7. Google Video

Безпечна спільна робота з відеоматеріалами без необхідності використання власних серверних ресурсів.

Google Apps для навчальних закладів містять безкоштовний (і вільний від реклами) набір інструментів, який дозволить викладачам і студентам більш успішно і ефективно взаємодіяти, навчати і навчатися. Пакет надає таки послуги:

- Спілкування - підвищує ефективність спілкування всередині навчального співтовариства за допомогою електронної пошти, календарів з загальним доступом і вбудованого відеочату.
- Спільна робота - Документи та Сайти Google дозволяють студентам і викладачам спільно створювати і редагувати документи в Інтернеті в будь-який час і з будь-якого місця.
- Інфраструктура та безпека - проста інтеграція існуючих IT-систем з Google Apps і забезпечення безпеки домену локальної установи.

Відкриті технології Cloud Computing

Існуючі технології «хмарних обчислень» дозволяють не тільки використовувати публічні сервіси як Amazon, але й самостійно розгортати «хмарні» інфраструктури як на базі підприємств, так і в домашніх умовах. Такі системи доступні кожному, і хоч їх установка наразі є непростим завданням, завдяки спільноті Open Sourse темпи розвитку відкритих проектів як Eucalyptus не поступаються гігантам Amazon EC2 та Microsoft Azure.

Eucalyptus - це розробка співробітників University of California, Santa Barbara, поширювана за ліцензією GPL v3. Eucalyptus надає відкриту реалізацію платформи хмари, що сумісна з форматом образів для Amazon EC2, таким чином обидві системи дуже просто порівняти та мігрувати з однієї на іншу.

Архітектурно Eucalyptus складається з декількох основних елементів:

- Cloud Controller - контролер «хмари», приймає запити від користувачів на створення віртуальних машин;
- Walrus - сховище для користувача даних і образів віртуальних машин;
- Cluster Controller - контролер кластера, розподіляє завдання із запуску віртуальних машин між вузлами;
- Node Controller - контролер вузла, управляє життєвим циклом віртуальних машин.

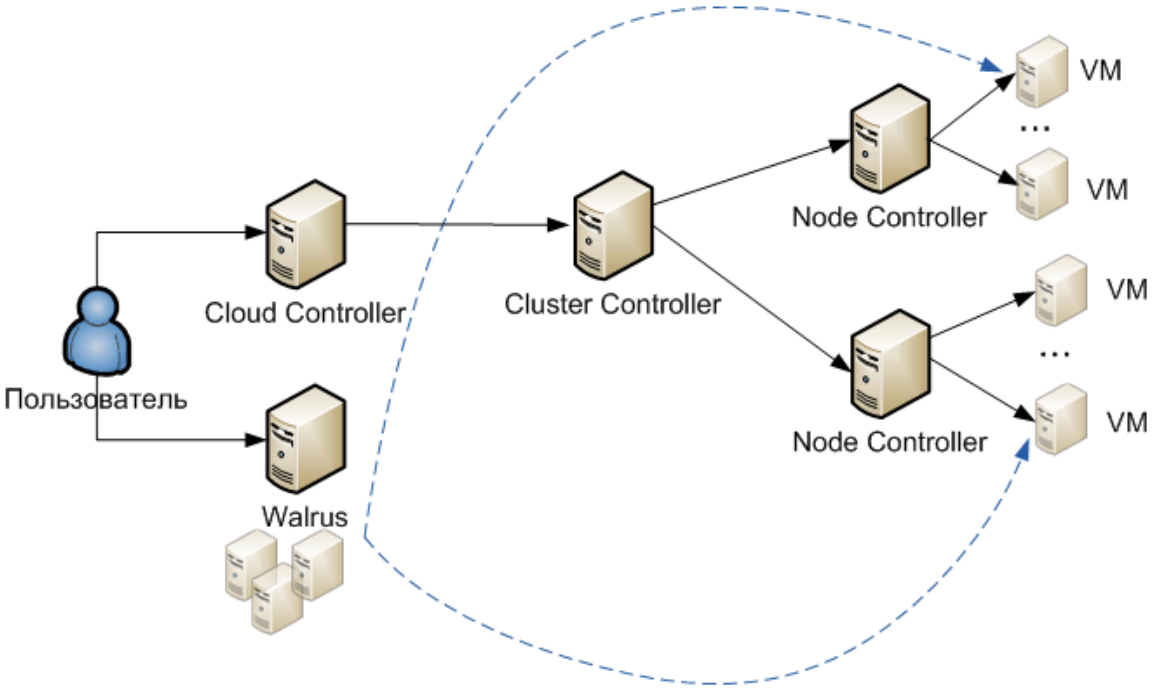


Рис. 7.1.9 Архітектура Eucalyptus

Технологічно EUCALYPTUS працює поверх кластеру під управлінням системи Rock Cluster і вимагає для розгортання платформи віртуалізації Xen 3.x. Система поширюється у вигляді образів для розгортання на всіх вузлах.

Для керування хмарою доступний web-інтерфейс, в якому можна роздавати права іншим користувачам, які будуть розміщувати свої віртуальні машини для виконання в середовищі хмари. Образи машин не просто сумісні з тими, що використовуються в Amazon, більш того їх можна створювати за допомогою тих самих інструментів. Таким чином Eucalyptus економить і свій час на створення власного формату образів та його підтримки, і час користувачів, адже їм не потрібно створювати різні образи однієї системи, що значно спрощує етап тестування та дозволяє легкий перехід між провайдерами хмарних обчислень.

Незважаючи на відносну молодість проекту і не повністю завершений функціонал, система вже представляє собою і дослідницький і комерційний інтерес, адже її цілком реально безкоштовно розгорнути на невеликому кластері і надавати послуги хостингу. Платформа може застосовуватися для надання як внутрішньокорпоративних, так і публічних сервісів. Варто зауважити, що всі компоненти можуть бути встановлені на один фізичний хост. Це накладає деякі обмеження на функціональність, але досить зручно для вивчення системи Eucalyptus.

Перспективи розвитку хмарних обчилень

Незважаючи на високу популярність хмарних сервісів, платформ та хостингів, хмарні обчислення тільки починають досягати піку інтенсивності свого розвитку. Інженери компаній як Amazon, Google та Microsoft постійно вдосконалюють свої платформи, розширяють API, додають все нові можливості та функції.

Однак, одноразово обравши свого провайдера, користувач стає назавжди прив’язаним до його специфічної платформи, адже ніякої стандартизації у

галузі хмарних обчислень досі не проводилося. Поки що тільки платформа Eucalyptus реалізувала сумісність образів з Amazon, але так як провайдери хмарних сервісів мають різні архітектури та пропонують різнорідні послуги, усім використовувати єдиний формат машин просто неможливо. Виходить, при необхідності змінити постачальника хмарних обчислень, клієнт вимушений нести високі збитки на переміщення свого прикладного програмного забезпечення із одної хмари на іншу.

Частково вирішити цю проблему покликаний проект FlexCloud, заснований у Технічному університеті Дрездену. FlexCloud орієнтується на хмарні сервіси наступного покоління, що передбачають більшу уніфікацію послуг хмарних провайдерів та автоматизований доступ до них через API. Розробкою такого гнучкого доступу до сервісів займається більшість провідних компаній, наприклад, Amazon починаючи з 2009 року дозволяє виконувати усі налаштування арендованої інфраструктури не тільки через web-інтерфейс, а і в реальному часі через API. Побудувавши бібліотеку, що реалізовуватиме деяку абстракцію поверх API різних провайдерів, FlexCloud пропонує себе у якості посередника між клієнтом та хмарною платформою, забезпечуючи моніторинг якості послуг провайдера. У разі невиконання угоди SLA, клієнт матиме можливість мігрувати своє прикладне програмне забезпечення на іншу сумісну “хмару” у автоматизованому режимі, уникаючи непередбачуваних витрат коштів та часу. Таким чином, FlexCloud захищає користувача від прив’язки до конкретної хмарної платформи, надаючи більшу свободу, надійність, та гарантуючи гнучкість у мігруванні між платформами.

Хмарні обчислення (англ. cloud computing) - технологія розподіленої обробки даних, в якій комп'ютерні ресурси і потужності надаються користувачеві як Інтернет-сервіс. Надання користувачеві послуг як Інтернет-сервіс є ключовим. Проте під Інтернет-сервісом не варто розуміти доступ до сервісу тільки через Інтернет, він може здійснюватися також і через звичайну локальну мережу з використанням веб-технологій.

Основою для створення і швидкого розвитку хмарних обчислювальних систем послужили великі інтернет-сервіси, такі як Google, Amazon і ін., а так само технічний прогрес, що по суті говорить про те, що поява хмарних обчислень була усього лише справою часу.

Види послуг, які надаються хмарними системами

Концепція хмарних обчислень припускає надання наступних типів послуг своїм користувачам:

- все як послуга (Everything as a Service);

При такому виді сервісу користувачеві буде надано все від програмно-апаратної частини і до управлінням бізнес-процесами, включаючи взаємодію між користувачами, від користувача потрібно тільки наявність доступу в мережу Інтернет.

- інфраструктура як сервіс (Infrastructure as a service);

Користувачеві надається комп'ютерна інфраструктура, зазвичай віртуальні платформи (комп'ютери), пов'язані в мережу.

- платформа як сервіс (Platform as a service);

Користувачеві надається комп'ютерна платформа зі встановленою операційною системою.

- програмне забезпечення як сервіс (Software as a service);

Цей вид послуги зазвичай позиціонується як "програмне забезпечення на вимогу", це програмне забезпечення розгорнуте на віддалених серверах і користувач може діставати до нього доступ за допомогою Інтернету, причому всі питання оновлення і ліцензій на це програмне забезпечення регулюється постачальником цієї послуги. Оплата в даному випадку робитися за фактичне використання програмного забезпечення.

- апаратне забезпечення як сервіс (Hardware as a Service);

В даному випадку користувачеві послуги надається устаткування, на правах оренди, яке він може використати для власних цілей. Цей варіант дозволяє економити на обслуговуванні цього устаткування, хоча за своєю суттю мало чим відрізняється від виду послуги "Інфраструктура як сервіс" за винятком того, що Ви маєте голе устаткування, на основі якого розгортаєте свою власну інфраструктуру з використанням найбільш відповідного програмного забезпечення.

- робоче місце як сервіс (Workplace as a Service);

В даному випадку компанія використовує хмарні обчислення для організації робочих місць своїх співробітників, налаштувавши і встановивши усе необхідне програмне забезпечення, необхідне для роботи персоналу.

- дані як сервіс (Data as a Service);

Основна ідея цього виду послуги полягає в тому, що користувачеві надається дисковий простір, який він може використати для зберігання великих об'ємів інформації.

- безпека як сервіс (Security as a Service).

Цей вид послуги надає можливість користувачам швидко розгортати, продукти дозволяють забезпечити безпечне використання веб-технологій, безпеку електронного листування, а також безпеку локальної системи, що дозволяє користувачам цього сервісу економити на розгортанні і підтримці своєї власної системи безпеки.

Види хмарних обчислень

З поняттям хмарних обчислень пов'язують сервіси, які мають (Everything as a service) технології, такі як:

• «**Інфраструктура як сервіс»** (" Infrastructure as a Service " або " IaaS ");
• **«Платформа як сервіс»** (" Plaatform as a Service ", " PaaS ");
• **«Програмне забезпечення як сервіс»** (" Software as a Service " або "SaaS ").

Інфраструктура як сервіс (IaaS)

IaaS – це надання комп'ютерної інфраструктури як послуги на основі концепції хмарних обчислень.

IaaS складається з трьох основних компонентів:

• Апаратні засоби (сервери, системи зберігання даних, клієнтські системи, мережеве обладнання);
• Операційні системи та системне ПЗ (засоби віртуалізації, автоматизації, основні засоби управління ресурсами);
• Сполучне ПЗ (наприклад, для управління системами).

IaaS заснована на технології віртуалізації, що дозволяє користувачу обладнання ділити його на частини, які відповідають поточним потребам бізнесу, тим самим збільшуючи ефективність використання наявних обчислювальних потужностей. Користувач (компанія або розробник ПЗ) повинен буде оплачувати всього лише реально необхідні йому для роботи серверний час, дисковий простір, мережеву пропускну спроможність та інші ресурси. Крім того, IaaS надає в розпорядження клієнта весь набір функцій керування в одній інтегрованої платформі.

Платформа як сервіс (PaaS)

PaaS – це надання інтегрованої платформи для розробки, тестування, розгортання і підтримки веб–додатків як послуги.

Для розгортання веб -додатків розробнику не потрібно купувати обладнання та програмне забезпечення, немає необхідності організовувати їх підтримку. Доступ для клієнта може бути організований на умовах оренди.

Такий підхід має такі переваги:
• масштабованість ;
• відмовостійкість ;
• віртуалізація ;
• безпека.

Масштабованість PaaS передбачає автоматичне виділення і звільнення необхідних ресурсів залежно від кількості обслуговуваних додатком користувачів. PaaS як інтегрована платформа для розробки, тестування, розгортання і підтримки веб-додатків дозволить весь перелік операцій з розробки, тестування, та розгортанню веб-додатків виконувати в одному інтегрованому середовищі, виключаючи тим самим витрати на підтримку окремих середовищ для окремих етапів. Здатність створювати вихідний код і надавати його в загальний доступ всередині команди розробки значно підвищує продуктивність по створенню додатків на основі PaaS.

Найвідомішим прикладом такої платформи є AppEngine від Google, яка пропонує хостинг для веб-додатків з можливістю купувати додаткові обчислювальні ресурси (наприклад, для тестування високих навантажень). Для запуску додатків Google AppEngine на віртуальних кластерних системах була розроблена платформа AppScale, яка не має ніякого відношення до Google.

У системах веб-пошуку і контекстної реклами компанії Yahoo використовується платформа Hadoop, орієнтована на передачу великих обсягів даних між мережевими серверами. На базі Hadoop побудовані HBase (аналог

бази даних Google BigTable), а також HDFS (Hadoop Distributed File System, аналог Google File System).

Ще одним яскравим представником PaaS є продукти компанії Mosso:
-Cloud Sites – веб-хостинг (Linux, Windows, Mail) для навантажувальних веб – проектів з можливістю розширювати базові безкоштовні – можливості за додаткову плату (трафік, сховище даних, обчислювальна потужність).
-Cloud Files – файловий cloud-хостинг з щомісячною погігабайтною оплатою за обсяг збережених файлів. Управління здійснюється через браузер, або за допомогою API (PHP, Python, Java, .NET, Ruby).
-Cloud Servers – погодинна оренда серверів (RAM на годину), з можливістю вибору серверної ОС. Можна змінювати характеристики сервера, але не в режимі реального часу.

Ну а в центрі всієї хмарної інфраструктури Microsoft – операційна система Windows Azure. Windows Azure створює єдине середовище, що включає хмарні аналоги серверних продуктів Microsoft (реляційна база даних SQL Azure, що є аналогом SQL Server, а також Exchange Online, SharePoint Online і Microsoft Dynamics CRM Online) і інструменти розробки (.NET Framework і Visual Studio, оснащена в версії 2010 набором Windows Azure Tools). Так, наприклад, програміст, який створює сайт в Visual Studio 2010, може не виходячи з програми розмістити свій сайт в Windows Azure.

Програмне забезпечення як сервіс (SaaS).

SaaS – модель розгортання програми, яка надає додаток кінцевому користувачеві як послугу на вимогу (on demand). Доступ до такого додатку здійснюється за допомогою мережі, а найчастіше за допомогою Інтернет-браузера. У даному випадку, основна перевага моделі SaaS для клієнта полягає у відсутності витрат, пов'язаних з установкою, оновленням і підтримкою працездатності обладнання та програмного забезпечення, що працює на ньому. Цільова аудиторія – кінцеві споживачі.

У моделі SaaS:

•додаток пристосований для віддаленого використання;
•одним додатком можуть користуватися декілька клієнтів;
•оплата за послугу стягується або як щомісячна абонентська плата, або на основі сумарного обсягу транзакцій;
•підтримка програми входить вже до складу оплати;
•модернізація програми може проводитися обслуговуючим персоналом плавно і прозоро для клієнтів.

З точки зору розробників програмного забезпечення, модель SaaS дозволить ефективно боротися з неліцензійним використанням програмного забезпечення, завдяки тому, що клієнт не може зберігати, копіювати і встановлювати програмне забезпечення.

Розвитком логіки SaaS є концепція **WaaS (Workplace as a Service – робоче місце як послуга).** Тобто клієнт отримує в своє розпорядження повністю оснащене всім необхідним для роботи віртуальне робоче місце.

Питання для самоконтролю

1.Охарактеризуйте програмне забезпечення хмарних обчислень.
2.Перерахуйте особливості програмування в «хмарах».
3. Що таке консолідація і віртуалізація IT-інфраструктури?
4. Що таке хмарні обчислення і на що націлені хмарні обчислення?
5. Назвати обчислювальні ресурси «хмар».
6. Назвати основні різновиди віртуалізації.
7. Що таке віртуалізація серверів (повна віртуалізація і паравіртуалізація)?
8. Що таке Віртуалізація на рівні ядра ОС (operating system – level virtualization)?
9. Що таке Віртуалізація додатків і Віртуалізація уявлень?
10. Що являє собою гіпервізорна технологія?
11. Охарактеризуйте архітектуру Hyper-v.
12. Охарактеризуйте Microsoft System Center Virtual Machine Manager (SCVMM).
13. Назвіть достоїнства та недоліки хмарних обчислень.
14. Перерахуйте види послуг, які надаються хмарними системами.
15. Що таке «Інфраструктура як сервіс» (" Infrastructure as a Service " або " IaaS ")?
16. Що таке «Платформа як сервіс» (" Plaatform as a Service ", " PaaS ")?
17. Що таке «Програмне забезпечення як сервіс» ("Software as a Service ", "SaaS")?

Література

1. Риз Дж. Облачные вычисления: Пер. с англ. — СПб.: БХВ-Петербург, 2011. — 288 с. ил.
2. Adobe Flash runtimes. Benefits. Режим доступу: http://www.adobe.com/products/flashruntimes/benefits.html
3.Windows Server 2008 R2 and Windows Server 2008. Режим доступу: www.microsoft.com/windowsserver2008/en/us/2008-dc. Aspx.
4. Crandell M. Defogging cloud computing: A taxonomy, June 16, 2008. Режим доступу: http://refresh.gigaom.com/2008/06/16/defogging-cloud-computing-a-taxonomy/
5. Google App Engine. Режим доступу: http://code.google.com/appengine Microsoft Azure. Режим доступу: http://www.microsoft.com/azure
6. http://www.pcweek.ru/themes/detail.php?ID=107230
7. http://www.ixbt.com/cm/virtualization.shtml

7.2 **Організація обчислень в хмарних середовищах**

Вперше мова про ідею хмарних обчислень пішла на конференції LinuxWorld 2003. Тоді звернули увагу на систему Scyld, яка була призначена для реалізації сайтів та бізнес-застосувань. Її особливістю було те, що будь-який потік запитів до сайту завжди обслуговувався за рахунок того, що призупинялись малопріоритетні процеси, а важливі процеси динамічно переміщувались на ресурси КК (комп'ют.класт.), які вивільняються. Відміною від ОС паралельних систем була та, що система Scyld могла виконувати

балансування навантаження великої кількості різних СЕ (процесор.елемент.), які мають власні ресурси та ОС різного типу.

Як і багато нових технологій, хмарні обчислення розвивались поетапно. На першому етапі хмарні обчислення розвивались як нова послуга – Software as a Service (SaaS), тобто програмне забезпечення як послуга в рамках Інтернет - стандарту Web 2.0. Почало з'являтись багато хостинг-сайтів, які були призначені для зберігання сайтів, веб-застосунків осіб та компаній. З'явились системи управління контентом (content management systems, CMS) і відповідно, традиційні сайти почали масово замінюватись на сайти на базі CMS. Тобто, тут CMS є типовим програмним забезпеченням як послуга на віддаленому сервері. Крім того, швидке зростання вартості електроенергії змусло багато фірм та організацій мінімізувати витрати, в тім рахунку використовуючи віддалений доступ до серверу.

На другому етапі відбулась віртуалізація хостингу. Віддалений доступ до серверів використовується вже не одне десятиліття і використовує, наприклад, протокол Microsoft Remote Desktop Protocol (RDP) та мережу VPN. Також давно застосовується віддалене тестування та налагодження серверів за допомогою запуску віртуального СЕ. При розвитку хмарних обчислень ця ідея набула подальшого розповсюдження у вигляді віртуалізації хостингу. При цьому віртуальний СЕ налаштовується разом з ОС та програмним забезпеченням у користувача, а потім цей СЕ разом з програмним забезпеченням перевстановлюється у віддаленому сервері.

Додатковий поштовх для розвитку віддаленого хостингу дали сплески цін на енергоносії у середині нульових років, через що тримати власний датацентр стало досить витратно. На цю тенденцію звернули увагу такі фірми, як Microsoft і Amazon, які почали будувати центри обробки даних в районах з джерелами недорогої енергії та з прохолодним кліматом, щоб мінімізувати витрати на живлення та охолодження центрів.

На третьому етапі були впроваджені дійсно хмарні обчислення. Хмарні середовища були результатом досвіду експлуатації Грід-систем та уроків віртуалізації хостингу. Цей досвід показав необхідність розробки галузевих стандартів для організації хмарних обчислень.

Хмарне середовище схоже як на Грід-систему, так і на КК (комп'ютерний кластер) у тому, що хмарні обчислення можуть бути організовані як у Грід-системі, так і у КК, якщо віртуальному Web-серверу користувача надати відповідні ресурси.

Тиск ринку змушує розвиток конкуруючих технологій хмарних обчислень стандартизувати ці технології. 27 жовтня 2008 р. фірма Microsoft першою представила свою версію хмарної ОС. Слід зауважити що хмарна ОС – це програмне забезпечення, яке дозволяє виконувати у хмарному середовищі один або декілька застосунків, які в свою чергу, дають змогу їх використовувати кінцевими користувачами.

Віртуальні процеси як основа хмарних обчислень

Реальні обчислювальні процеси у системі обробки даних, як правило, простоюють до 80-90% часу, очікуючи готовність даних. За рахунок віртуалізації можна запустити кілька обчислювальних процесів на одному реальному СЕ, завдяки чому підвищити ефективність використання системи. Крім того, якщо ця віртуалізація виконується в Грід-системі, або КК, то досягається економія енергоспоживання, оскільки у цих системах ефективність джерел живлення досягає 95% . Крім того кожен віртуальний процес можна зупинити, зберегти, переслати і поновити виконання у будь-якому місці хмарного середовища.

Одним з головних факторів, які ускладнюють організацію обчислень у хмарному середовищі, є відсутність ефективних засобів для переміщення застосування у хмарне середовище. Досі процес автоматичного завантаження завдання у хмару реалізується за допомогою набору утиліт від несумісних систем (Microsoft, Amazon, Google і т.д.). З боку користувача існують програми, які часто некоректно називаються хмарними ОС, які допомагають переслати дані та команди у хмарне середовище. Це, наприклад, системи Dropbox і Microsoft Mesh. Але на стороні сервера все набагато складніше через проблеми, які виникають при віртуалізації процесів.

Віртуалізація процесу пов’язана з його абстрагуванням. Тут під абстрагуванням мається на увазі визначення місцезнаходження процесу серед рівнів обчислювальної системи (від рівнів апаратури, драйверів, ОС до рівня користувацької програми) та визначення механізмів взаємодії з нижчерозташованим та вищерозташованим рівнями. Наприклад, рівень .NET, який додано у ОС Windows, успішно сприяє розробці мережевих застосунків, бо він приховує від розробника усі рівні, що лежать нижче, тобто абстрагується від них. Такий самий рівень необхідний для реалізації хмарних обчислень. Тоді відпадає необхідність для користувачів і адміністраторів явно визначати СЕ, на якому буде виконуватись конкретний застосунок.

Для віртуалізації у хмарному середовищі використовується рівень гіпервізора. Коротко ми це розглянули в попередній лекції, але вона була досить об’ємна і тому там я нічого не сказав про Windows Server 2012, плануючи присвятити цьому питанню окрему лекцію. Але так як кількість лекцій обмежена (їх має бути 18), коротко зупинюсь на цьому питанню в цій лекції.

Гіпервізор (ще називається монітор віртуального СЕ, virtual machine monitor, VMM) – це програмна або апаратно-програмна платформа для віртуалізації програмного забезпечення, яка дає змогу кільком ОС одночасно виконуватись на одному фізичному СЕ. Наприклад, гіпервізор Windows Server 2008 R2 Datacenter дає права на запуск необмеженої кількості ОС Windows Server на сервері.[1]

Можливості Hyper-V в сімействі операційнх систем Windows Server 2008 вивели Microsoft на гідне місце на ринку віртуальних серверів, однако Hyper-V все ще був на крок позаду свого основного конкурента, VMware, в відношенні масштабованості, високої доступності та функціональності. В Windows Server 2012 Microsoft догнала VMware і в цих питаннях. Розробкою Hyper-V

програмісти Microsoft заклали основу для інтегрованої технології віртуалізації серверів прямо в базовій операційній системі Windows Server. Базова технологія, відома як гіпервізор (hypervisor), по суті являє собою рівень в середині головної операційної системи, яким забезпечує кращу підтримку гостьових операційних систем. Як вже було сказано, свою гіпервізорну технологію компанія Microsoft назвала Hyper-V. До включення Hyper-V в Windows Server 2008 і Windows Server 2012 стек віртуалізації розміщувався поверх головної операційної системи і, по суті, вимагав, щоб всі гостьові операційні системи спільно використовували системні ресурси, такі як мережеві комунікаційні ресурси, можливості відеообробки та виділення памяті. При системному збою в головній операційній системі, наприклад, відмові драйверу мережевого адаптеру, всі гостьові сеанси втрачають можливості обміну даними по мережі.

Технології, подібні VMware ESX, Citrix XenServer і Hyper-V збільшують ефективність технологій, побудованих на основі гіпервізору, дозволяючи гостьовим операційним системам ефективно обходити головну операційну систему і безпосредньо обмінюватися даними з системними ресурсами. В одних випадках гіпервізор керує спільно використовуємими ресурсами гостьового сеансу, а в інших випадках гостьові сеанси оминають гіпервізор і відправляють запити безпосередньо на апаратний рівень системи. Забезпечуючи більшу незалежність обміну даними між системами, середовище з гіпервізором предоставляє організаціям більшу степінь масштабованості, більш високу продуктивність і більш високу надійність базового середовища віртуального хосту.

В Windows Server 2012 технологія Hyper-V підтримується тільки в 64-розрядних системах з апаратною підтримкою віртуалізації. ЦП має підтримувати опцію Intel VT, або AMD-V і режим захисту від виконання даних (Data Execution Protection — DEP). Крім того, ці функціональні можливості повинні бути включені в BIOS комп'ютера.[2]

Hyper-V в Windows Server 2012 містить значні покращення, які зробили Hyper-V не тільки достойним товаром на ринку віртуалізації серверів, але і лідером, який підняв планку вимог до технологій віртуалізації серверів в організаціях. Він містить багато довгоочікуваних компонентів і технологій, які можна розбити на три великі категорії: збільшена потужність хосту і гостьових сеансів, інтегровані технології високої доступності і покращена керованість. В Hyper-V є можливість розміщення гостьових операційних систем на серверах Windows, кліентських системах і системах з іншими ОС. А всякі інструменти і керування віртуальними хостами і гостьовими сеансами підсилюються використанням диспетчеру віртуальних машин (Virtual Machine Manager — VMM) із системного центру Microsoft System Center 2012. VMM позволяє створювати відображення фізичного серверу на віртуальний і копіювати віртуальний сервер на віртуальний та розширює поняття керування з віртуальних гостів і хостів на всю мережеву "фабрику". Фабрика мережі включає управління SAN, створення віртуальних локальних мереж (virtual local-area network — VLAN) і автоматичне створення повних дво- і трьохрівневих зв'язок серверів.

Центр керування системою System Center 2012 доповнює можливості управління і адміністрування хостами Hyper-V. Весь пакет продуктів System Center 2012 описаний в книзі System Center 2012 Unleashed, яка присвячена не тільки використанню VMM для автоматизації процесів в ускладнюючихся середовищах гостьових сеансів, але і інших засобів: диспетчер конфігурації для виправлення і керування серверами-хостами і віртуальними гостьовими сеансами, диспетчер захисту даних для архівації хостів і гостьових серверів і інші компоненти їх семейства System Center.

Іншою прогресивною технологією, яка поширюється разом з віртуалізацією, є технологія мінімізації програмного забезпечення, яке виконується. Замість повної версії ОС для кожного віртуального СЕ, серед цих СЕ поширюється лише ядро ОС та необхідні додатки до нього, а загальні додатки розділяються серед кількох віртуальних СЕ. Завдяки цьому скоротились витрати на пересилки програм.

Іншим важливим аспектом є керування віртуальним СЕ. Можливість змонтувати образ диску, породжуючи новий екземпляр ОС з диску віртуального образу ОС, з'явилась майже одночасно в системах управління для блейд-серверів і у консольних застосунках віртуальних машин. Ця можливість дала змогу віддалено керувати віртуальним СЕ та виконувати автоматизоване балансування навантаження.

У той же час, можливість запуску віддаленої консолі для кожного віртуального СЕ з браузеру дала змогу продовжувати керування цим СЕ після того, як на нього встановлена ОС. Але зростаюча залежність від рівнів абстракції і розмноження екземплярів ОС призвело до наступної проблеми. Хоча рівень абстракції призначений для фізичного розділення нижнього рівня від верхнього, але ці рівні виявляються неспроможними бути цілком розділеними, бо система перестане функціонувати.

У той час, як віддалені консольні застосунки були вже розроблені і впроваджені, програми моніторингу мережі, такі як HP Open View, OpenNMS, Пакет Trap, Nagios, і UniCenter тільки почали використовуватись для аналізу віртуалізованого середовища. Така програма моніторингу може забезпечити точну інформацію в реальному часі, наприклад, про те, коли апаратні ресурси можуть бути перевантаженими, або коли тривалість ліцензії добігає до кінця.

Завдяки таким програмам та віртуалізаціїї процесів, у одній стійці сервера можна реалізувати сотні і тисячі віртуальних ОС. Помітною подією була поява блейд-серверів, або систем 234 (модульні, розроблені в 2001р, займають в 2 рази менше місця, споживання в три рази менше енергії, дешевші в 4 рази) серії С3000 фірми Hewlett-Packard. Такий сервер набирається з компактних процесорів та пакетів жорстких дисків у металевих корпусах (блейдах), які можуть вийматись та встромлятись у задню панель стійки без порушення роботи сервера. При цьому у блейдах, які щойно підключені до живлення та швидкодіючої локальної мережі, розміщується необхідна кількість нових віртуальних СЕ.

Поява та удосконалення хмарних обчислень

Отже, поява рівня абстракції віртуальної ОС, механізму віртуалізації та можливості переходу віртуального СЕ з одного фізичного СЕ на інший були

передумовою для появи хмарних обчислень. Виникає питання, що відрізняє хмарні обчислення від віртуалізації? Спочатку вважалось, що відміна у розташуванні серверів та комп'ютерів користувача. Насправді, віртуалізація означає встановлення екземплярів віртуальних ОС на серверах, що належать організації користувача. А хмарні обчислення надаються на серверах, розташованих у "хмарі" Інтернету, в місцях, які навіть не відомі користувачу. Тобто, обчислювальні ресурси, які виділяються, не мають конкретного місця і є фізично недосяжними для користувача.

Отже, за визначенням, ***хмарні обчислення*** – це набір комп'ютерних послуг, які надаються виключно в якості служби, без відомостей про конкретний сервер, який замовляється споживачем. Провайдер хмарного сервісу (ПХС, CSP) має забезпечити встановлення віртуальних ОС будь-якого типу. Ефективність ПХС залежить від диспетчера завдань (scheduler), який складається з брокера та агента.[3]

Брокер надає інтерфейс користувача для зовнішніх процесів, щоб спілкуватись з хмарним застосунком, а агент – це програма, яка сканує хмарні сервіси, щоб знайти дані для застосунку користувача. Наприклад, брокер – це вікно браузера, а агент – засіб гортання сторінок Веб-магазину. Разом ці дві частини планувальника узгоджують рівні абстракції і компонентів, щоб організувати виконання застосунку у хмарі. Таке планування дозволяє підвищити рівень абстракції компонентів, тобто тепер застосунок можна створити з великої кількості розрізнених компонентів, навіть розміщених у різних ПХС.

Іншою ознакою хмарних обчислень є концепція групової ідентифікації (*identity federation*). При груповій ідентифікації користувачу досить зареєструватись у першої організації-постачальника послуг, що дає право не реєструватись у інших організацій, бо вони можуть перевірити права користувача, звернувшись до першої організації. Наприклад, зареєструвавшись в Facebook, користувач може не реєструватись на інших сайтах.

Ще однією особливістю хмарних обчислень є служба з'єднання (cloud dating service). Це засіб перевірки прав користувача третьою стороною. Наприклад, користувач починає підключення до веб-сайту, який в свою чергу починає процедуру створення захищеного з'єднання. Цей сайт має набір облікових даних, які надсилаються на веб-браузер користувача. Браузер, у свою чергу, посилає ці зашифровані облікові дані на сторонній сервер сертифікації. Після успішної сертифікації веб-сайт дає відповідний дозвіл для роботи користувача.

У хмарному середовищі часто застосовується також самоадміністрування. Концепція віртуальних машин, які самоадмініструються, основана на можливості автоматичного породження нових екземплярів віртуальних СЕ при необхідності збільшення продуктивності для виконання завдання. Такі можливості, наприклад, закладені в ОС VMotion фірми VMWare.

Користувачі хмарного середовища повинні мати можливість розробляти або налаштовувати додаток, протестувати його на локальних ресурсах, а потім через агента передати його постачальникам хмарних послуг. Великою

перевагою хмарного середовища є можливість "взяти в оренду" застосунки, розроблені іншими, для того, щоб швидше задовольнити свої нові потреби.
Новий вид послуг, коли береться в аренду стороннє програмне забезпечення, надається рядом постачальників, у тім рахунку Amazon, Google і IBM. Для розвитку цього напрямку необхідно створити стандартний користувацький інтерфейс та стандартну платформу (додатковий рівень абстракції) для виконання застосунків, яка не залежить від рівня ОС, який знаходиться нижче.
Одним з перших прикладів того, як має виглядати середовище для розробки хмарних застосунків, є середовище Adobe Integrated Runtime (AIR). AIR має частини, які необхідні для кожної системи розробки для хмарних обчислень, в тім рахунку:
• дескриптори запитів ресурсів у форматі XML;
• можливість підтримки декількох об'єктно-орієнтованих систем Веб-розробки (Java, Flash);
• можливість підтримки інтерфейсів Інтернету, баз даних і апаратних інтерфейсів, які не залежать від типу ОС (Windows чи Linux).

Класифікація хмарних систем

На сьогодні прийнято три різних класифікації видів хмарних обчислень, які з'явилися в різний час еволюції у цій комп'ютерній сфері.

1 SPI - класифікація

Протягом перших впроваджень хмарних обчислень вони зразу були розділені на три основних категорії: програмне забезпечення як послуга (Software as a Service, SaaS), платформа як послуга (Platform as a Service, PaaS), і інфраструктура як послуга (Infrastructure as a Service, IaaS). За першими буквами цих категорій ця класифікація й одержала назву SPI Розглянемо їх конкретніше.
SaaS – це підмножина хмарних систем, яка призначена для виконання застосунків, створених для хмари і розгорнутих у хмарі, які знаходиться в Інтернеті. Цільовий користувач SaaS є кінцевим користувачем. Ці застосунки – хмарні застосунки, як правило, побудовані на основі браузера і мають наперед визначені функціональність та можливості. Прикладом SaaS є застосунки Google, такі як Google Docs і Google Spreadsheets.
SaaS вважається привабливою альтернативою застосункам на ПК. Наприклад, маючи застосунок, розгорнутий на сервері провайдера, зменшуються вимоги до обладнання ПК, немає проблем з технічним обслуговуванням, чи з модернізацією програмного забезпечення.
В випадку PaaS провайдер забезпечує наявність надійної програмної платформи з інтерфейсом прикладного програмування (API), яка може бути використана в розробці хмарних застосунків. Одним із прикладів систем в цій категорії є платформа Google App Engine, яка забезпечує середовище і API-інтерфейси для виконання програм на мовах Python і Java, призначених для сфери Google.[4] Microsoft Azure також відноситься до PaaS.[5]
Розробка програми для хмарної платформи подібна до розробки веб-застосунку для веб-сервера в тому сенсі, що результуюча програма буде розгорнута на віддаленому сервері. Однак модель PaaS відрізняється тим, що платформа може

надавати додаткові послуги, щоб спростити розробку програм, їх розгортання і виконання, наприклад, автоматичного масштабування, моніторингу та балансування навантаження, спрощення програмування таких функцій, як служб аутентифікації, служб електронної пошти і компонентів інтерфейсу користувача.

Крім того, розробники програми можуть інтегрувати інші послуги, які надаються системою PaaS для їх застосування, наприклад, служб аутентифікації, служб електронної пошти і компонентів для користувача інтерфейсу, механізмів вимірювання завантаження, чи організації платежу за послуги. Все, що забезпечується за рахунок набору API, поставляється з платформи. В результаті, клас PaaS, як правило, призначений, щоб прискорити розробку програмного забезпечення і час розгортання. В свою чергу, програмне забезпечення, яке побудовано для хмарної платформи, зазвичай має більш короткий час виходу на ринок. Деякі наукові проекти з'явилися також на підтримку більш повного розуміння PaaS, таких як AppScale.

Клас систем IaaS забезпечує користувачів ресурсами інфраструктури, такими як обчислювальні ресурси, зберігання даних, послуги зв'язку. Найбільш відомий приклад – це Elastic Compute Cloud (EC2) фірми Amazon.[6]

Досягнення в області віртуалізації ОС сприяли реалізації IaaS на існуючих серверах. В результаті, віртуалізації ОС є можливість відноситись до усього програмного забезпечення та пов'язаних з ним ресурсів окремого користувача як до об’єкту, який можна призначити на фізичні ресурси, які знаходяться будь-де і виділити йому відповідну частку робочого часу. Тому віртуалізація ОС дає змогу провайдерам IaaS керувати ефективним використанням фізичних ресурсів при збереженні знайомого інтерфейсу до стандартних апаратних платформ і мереж з використанням існуючої практики створення і використання програмного забезпечення.

2 Класифікація UCSB-IBM

Ця класифікація була виконана при співробітництві між Каліфорнійським університетом в Санта-Барбарі (UCSB) та дослідним центром фірми IBM. У цій класифікації розглядаються взаємовідносини різних рівнів хмарного середовища за принципом можливості рівнів об’єднуватись для реалізації складеного сервісу (composability), що запозичено з класифікації сервісно-орієнтованих архітектур (SOA). Тоді хмарний сервіс – це композиція одного чи більше інших сервісів.

Модель UCSB-IBM складена з п’яти рівнів відповідно до цього принципу. Кожен рівень охоплює один або кілька хмарних сервісів. Хмарні сервіси відносяться до одного рівня, якщо вони мають однаковий рівень абстракції. Наприклад, всі хмарні середовища (хмарні платформи) орієнтовані на програмістів, а хмарні застосунки – на користувачів. Тому перші та другі належать до різних рівнів. У моделі UCSB-IBM рівні формують стек хмари, причому один рівень знаходиться нижче у стеку, якщо він забезпечує виконання послуг у рівні, який вище. Графічно ця модель показана на рис.3.2.1.

Рис.3.2.1. – Класифікація хмарних обчислень UCSB-IBM

Перші три рівні класифікації хмарних обчислень UCSB-IBM аналогічні класифікації SPI, за винятком того, що рівень інфраструктури розділено на три компоненти: обчислювальні ресурси, пам’ять та накопичувачі даних і зв'язок.
Перший – верхній рівень – це рівень застосунків (SaaS). Як правило, користувачі отримують доступ до послуг цього шару через браузер. Практикою доведено, що ця модель приваблива для багатьох користувачів, оскільки це полегшує обслуговування програмного забезпечення і зменшує допоміжні витрати. Крім того, вона забезпечує передачу обчислювальної роботи від терміналу до центрів обробки даних, що зменшує вимоги до обладнання, дозволяє отримати високу продуктивність без великих капіталовкладень.

Завдяки поширенню послуги SaaS відбувається зростання нового класу пристроїв користувача у вигляді нетбуків, тобто дешевих пристроїв кінцевих користувачів з невеликим розміром пам’яті та малопотужним процесором, робота яких основана на мережевих підключеннях і хмарних застосунках.
Що стосується постачальників хмарних послуг, ця модель спрощує їх роботу по відношенню до модернізації та тестування коду, захисту інтелектуальної власності.

Хмарні застосунки розробляються з використанням хмарного програмного забезпечення та компонентів інфраструктури. Крім того, хмарні застосунки можуть компонуватися як сервіс від інших служб з використанням понять SOA. В цьому відношенні, хмарні застосунки, призначені для вищих рівнів, стають простіше в розробці, в них менше помилок, так як всі їх взаємодії з хмарою виконуються через перевірений програмний інтерфейс. Але безпека і доступ до хмарних застосунків є основним питанням у цій моделі. Крім того, невирішеною проблемою залишається боротьба з простоями ресурсів у хмарі, включаючи простої від відмов мережі та СЕ. Крім того, інтеграція множини застосунків і переміщення даних користувача в хмару гальмують поширення моделі SaaS, що пов’язане з побоюваннями за ненадійність зберігання конфіденційних даних, аутентифікації і авторизації, безвідмовної роботи і продуктивності, резервного копіювання і відновлення після збоїв.

Другий рівень – це рівень програмного забезпечення або програмної платформи. Користувачами цього рівня є розробники хмарних застосунків. Провайдери забезпечують розробників відповідними засобами програмування

та чітко визначеним інтерфейсом API. Таку послугу називають платформою як сервіс (PaaS). Прикладами такої платформи є система з мовою Apex фірми Salesforce, що дозволяє розробникам створювати крім власне програм макет сторінки, об'єднання модулів і звіти клієнтів.[7] *PaaS* також забезпечує автоматичне масштабування і балансування навантаження, а також інтеграцію з іншими послугами (наприклад, послугами аутентифікації, електронної пошти та створення інтерфейсу користувача). Отже, більша частина накладних витрат розробки хмарних застосунків полегшується і оброблюється на рівні хмарного програмного забезпечення. Це спрощує і прискорює розробку хмарних застосунківння, і зводить до мінімуму кількість помилок. Для цього розроблюються спеціальні програмні комплекси, такі як Map Reduce [8], Hadoop [9].

Третій рівень – це рівень інфраструктури хмарного програмного забезпечення, який складається з трьох частин: обчислювальних ресурсів, системи зберігання даних та телекомунікаційної системи.

Віртуальні СЕ – це загальна форма надання обчислювальних ресурсів для користувачів в цьому шарі. Віртуалізація ОС дає користувачам гнучкість у налаштуванні їх параметрів, захищаючи фізичну інфраструктуру серверів провайдера. Користувачі зазвичай отримують доступ суперкористувача, щоб налаштувати програмний стек на їх віртуальних СЕ для підвищення продуктивності та ефективності. Тому такі послуги назвали інфраструктура як сервіс – IaaS.

Ресурс інфраструктури зберігання даних – DaaS – дає змогу користувачам зберігати свої дані на віддалених дисках і отримати до них доступ в будь-який час з будь-якого місця і це полегшує масштабування хмарних застосунків. Прикладом комерційної хмарної Daas системи є S3 фірми Amazon [10].

Телекомунікації – це життєво важливий компонент хмарної інфраструктури, якість якого забезпечує якість усього хмарного сервісу. Тому заради досягнення високого рівня цієї якості виділяється в окрему категорію телекомунікації як послуга (CaaS), які між іншим повинні відповідати вимогам мережевої безпеки, динамічного виділення віртуальних каналів для ізоляції трафіку, регулювання трафіку, гарантовану межу затримки повідомлення, шифрування зв'язку та моніторингу мережі.

Рівень ОС та проміжного програмного забезпечення забезпечує основні функції управління програмним забезпеченням для фізичних серверів, які формують хмару. Цей рівень може бути реалізований у вигляді ядра ОС, гіпервізору, віртуальних СЕ, монітору і / або кластеризації проміжного програмного забезпечення. Фактично – це рівень, на якому стоїть рівень застосунків Грід-систем. Системи Globus[11] і Condor[12] є двома успішними системами проміжного програмного забезпечення для Грід-систем. Зараз багато концепцій, які були розроблені для Грід-систем, використовуються у хмарних обчисленнях. У подальшому можливе запозичення нових розробок проміжного програмного забезпечення хмар для Грід-систем.

Апаратне забезпечення та вбудоване програмне забезпечення формують нижній рівень стека рівнів хмарних обчислень в класифікації UCSB-IBM.

У класифікації UCSB-IBM відсутній аспект безпеки хмарних обчислень. З акцентом на підтримку хмарних обчислень для урядових установ у роботі[13] представлена адаптована класифікація UCSB-IBM.

Рівень управління доступом додається над рівнем хмарних застосунків і призначений для забезпечення керування доступом до програм, що реалізують SaaS. У вигляді різних методів аутентифікації, цей рівень може забезпечити спрощену і уніфіковану, але ефективну форму захисту. В свою чергу, це може спростити розробку і використання додатків SaaS з вирішенням проблем безпеки для цих систем.

3 Модель хмари Хоффа

Під впливом перших двох класифікацій хмарних обчислень Крістофером Хоффом була розроблена більш розширена класифікація у вигляді моделі хмарних обчислень [14,15] .

У цій моделі увага приділяється аналізу трьох основних хмарних сервісів: IaaS, PaaS і SaaS. У моделі рівень IaaS складається з кількох компонентів, перший з них характеризується продуктивністю і простором (ємністю пам'яті), дугий – це обладнання, яке складається з набору CE, сховища даних та мережевої системи. Третій компонент – засоби абстрагування, такі як монітор віртуальних CE, утиліти звернення до Грід-системи, кластеру, а також засоби мережевого з'єднання та постачання послуг, які забезпечують різні сервіси, наприклад, аутентифікація, DNS. Компонент API представляє послуги з управління, а також спрощений інтерфейс до наступного рівня в хмарі. Наступний рівень в моделі Хоффа, який є PaaS, складається з одного підрівня, який надає послуги з інтеграції в хмарі і забезпечує декілька послуг, таких як аутентифікація, організація баз даних і запитів до них.

Рівень SaaS також розбитий на кілька підрівнів і компонентів. Підрівень даних означає фактичні дані, метадані і їх зміст, який може бути у структурованій або неструктурованій формі. Прикладний компонент в рівні SaaS поділяється на три категорії: стандартні застосунки, веб-застосунки та вбудовані застосунки. Останні два підрівні у рівні SaaS – це API додатків та підрівень застосунків для відображення даних, який далі розкладається на відображення даних, аудіо- відеоданих, тобто різних форм представлення даних у хмарі.

Класифікація Хоффа відображає більш детальну інформацію про склад хмарного середовища, яка показує додаткові аспекти та особливості хмарних обчислень. Вона доповнює та уточнює дві попередні класифікації. Онтологічний підхід, задіяний в двох останніх класифікаціях дає змогу порівнювати існуючі та нові хмарні середовища, ототожнюючи більш ефективно їх нові аспекти та надихаючи на створення нових більш ефективних сценаріїв використання хмари, характеристики в світлі їх залежності від інших обчислювальних полів і моделей.

Розуміння компонентів хмарного середовища дає змогу знайти нові рішення загальних проблем хмарних обчислень, таких як доступність, міграція застосунків між хмарними середовищами, стійкість системи до відмов, хакерських атак, тощо.

Розробка застосунків для хмарного середовища

1. Переваги веб-браузеру як інтерфейсу користувача

Хмарні застосунки повинні мати функції, які розділені на частини: програми, які виконуються на боці користувача і програми, які виконуються на сервері. Але у випадку хмарних обчислень, частина користувацького інтерфейсу, як правило, ближче до користувача, і становить основну частину, яка потребує розробки.

Поширена помилка, що хмарні застосунки повинні по формі нагадувати традиційні веб-сайти. Однак, коли почали з'являтися нові середовища розробки, такі як Adobe AIR, стала розмиватись традиційна відміна між застосунками, які керуються з командного рядка та застосунками, які керуються з веб-браузера. Нова тенденція полягає в тому, що веб-застосунки не повинні бути обмежені межами веб-браузера.

Веб-браузер став зручним способом для виконання багатьох застосунків, оскільки він дозволяє просте розгортання у різних ОС і спрощене обслуговування застосунків. Крім того, сучасні мови програмування, які використані в браузері, дозволяють швидко розробити застосунки, які зв’язуються з віддаленим сервером. Так, середовище Adobe AIR доповнює браузер, надаючи ті самі переваги розробки застосунку, як і при розробці звичайної програми для РКС (розподіл. сист.), додаючи можливості багатофункціональних інтернет-застосунків (Rich Internet Applications, RIA) [16] .

Програмування функцій браузера з самого початку непросте. Наприклад, серйозною задачею є виділення контексту, на який вказує курсор на екрані. Але тепер функції браузера так тісно інтегровані в ОС, що стало можливим звертання до функцій браузера за допомогою операторів мови програмування. Тепер ключовими компонентами застосунків стають функції відеоконференцій та плагіни, такі як Enterprise Connect від Adobe, WebEx, або Live Meeting від Microsoft.

Регулярна архітектура платформи браузера несе ряд переваг і кілька суттєвих ризиків. Серед переваг є просте підключення функцій, плагінів, невеликий об’єм необхідної пам'яті та ресурсу СЕ для них, просте розширення функцій. Недолік, як правило, пов'язаний з безпекою, так як користувачі часто додають нові функції до браузера, не завжди усвідомлюючи наслідки цього.

2. Плагіни та веб-платформи

Поведінка і вплив плагінів ускладнюється тим, що деякі плагіни діють без втручання користувача і можуть сильно впливати на роботу застосунку, який розробляється. Наприклад, відомо, що при виборі плагінів певних панелей інструментів, змінюються домашні сторінки, вибір інших додатків і т.д.

Багато користувачів не знайомляться детально з призначенням плагінів. В деяких випадках використання плагінів неусвідомлено призводить до протизаконного поводження з інформацією. Хоча є правильно розроблені плагіни, які не виконують побічних дій, займають мало місця і самостійно видаляються з комп’ютеру, якщо в них немає необхідності.

При цьому слід відмітити революційний вплив появи таких платформ, як Ruby On Rails, Flex і Ajax, які служать для абстрагування на рівні програмування і які призвели до істотного зрушення в сутності програм, які називаються застосунками. Все, що виконують ці системи, також може бути зроблено на мові нижчого рівня, тобто, застосовуючи PERL або PHP замість Ruby On Rails. Але прийдеться витратити значно більше часу для розробки та налагодження такої програми.

Ще однією перевагою програмування на рівні абстракції є те, що ці системи, як правило, стандартизовані, що також збільшує можливість повторного використання коду.

Недоліком застосування веб-платформ є те, що втрачається швидкість роботи. Для того, щоб відреагувати на будь-яку потенційну ситуацію, веб-платформа повинна мати велику бібліотеку функцій, щоб охопити більшість ситуацій. Так, видача повідомлення на екран може бути закодована одним рядком на такій мові, як Python, але він розширюється до декількох сотень або навіть тисяч рядків коду під час їх інтерпретації, коли викликаються відповідні бібліотечні функції. Отже, таке гальмування виконання веб-функцій є передумовою необхідності постійного зростання продуктивності комп'ютерів, в тому рахунку є виправдовуванням застосування закону Мура.

3. Низькорівневі та високорівневі мови програмування

Вважається, що найшвидкіші програми слід складати на мовах низького рівня, тому, що компілятор такої мови не здатен додавати надлишок команд, необхідних для запобігання усіх можливих випадків. Тому набагато легше налаштувати таку програму при жорстких часових обмеженнях, тому що в ній немає кодів, які необхідно мінімізувати.

Мови ассемблера та С і раніше були найпопулярнішими мовами програмування для систем керування і драйверів пристроїв через їх надзвичайно короткий код. Але недоліком є те, що такі короткі програми важко складати і праця такого програміста є високовартісною. Тому не дивно, що такі рівні абстракції, як Ruby, розроблялись на мовах Rails, Python, SPSS, SAS і т.д. Так що верхні рівні абстракції спрощують поєднання програмних блоків один з одним.

Прикладом ефективного програмного середовища для розробки застосунків, яке представляє високий рівень абстракції, є Adobe AIR (Adobe Integrated Runtime). AIR дозволяє розробникам писати програми, які абстрагується від основної ОС і апаратного забезпечення. Ця платформа забезпечує проміжний шар, який створює ілюзію у Java- чи Flash-програмістів, що вони працюють над ОС Microsoft Windows або Apple Macintosh. AIR також дає інструменти для розробки в галузях HTML / AJAX, Adobe Flash і Flex.

Аналогічним середовищем є Microsoft .NET, яке дає змогу суттєво зменшити роботу, необхідну для отримання доволі складних систем. Але .NET обмежене ОС сімейства Microsoft, ігноруючи той факт, що основна частина веб-серверів у світі побудована на Apache, велика частина яких встановлена на ОС Linux. Середовидще Microsoft Silverlight, хоча має гірші властивості, ніж AIR, але дає змогу маніпулювати Web-даними, такими, які забезпечуються послугами Netflix відео за запитом.

Отже, низькорівневі мови програмування – це не для Веб-застосунків. Вони використовуються для розробки середовищ, які підвищують рівень абстракції програмування користувача, завдяки чому таке програмування спрощується, а самі застосунки можуть бути переміщувані і незалежні як від апаратури, так і ОС.

4. Бази даних в Інтернеті

Програмування веб-застосунків потребує стандартний спосіб доступу баз даних. Для цього був створений рівень абстракції, такий як ODBC (Open Database Connectivity), який став стандартним для кількох ОС. Іншим чинником, який дозволяє звертатись до хмарної бази даних, є загальний інтерфейс шлюзу (Common Gateway Interface, CGI), створений у 1993 р. і ставший основою для великої кількості інших засобів, призначених для створення веб-застосунків. Цей інтерфейс стандартизує методологію для зв'язку програм за межами веб-сервера, так що складні застосунки можуть бути пов'язані з базами даних у веб-середовищі.

5. Хмарне середовище

Хмарне середовище –це найвищий рівень абстракції, який приховує системні деталі від кінцевих користувачів. Існує тенденція, що хмарне середовище будується таким чином, що користувач не бачить ОС, але бачить велику бібліотеку застосунків, з якої вибирає для себе потрібні з оплатою їх використання.

Для Вас у хмарному середовищі найцікавішим є рівень SaaS. Але розробити, відлагодити та розмістити застосунки у хмарному середовищі – цього замало для успішної реалізації програмного проекту. Для досягнення успіху бажано дотримуватись наступних рекомендацій .[17]

Слід навести зв’язки з публічними мережами. Якщо буде посилання на SaaS в публічній мережі, такий як LinkedIn, Фейсбук, то більша ймовірність того, що застосунок побачать і знайдуть ті, кому він потрібен. При цьому його можна подати з різними ухилами: навчальний, науковий, технічний, соціальний. Головне, щоб його контент був: оригінальним, корисним, оптимізованим для серверів-пошуковиків.

Бажано на початку розповсюдження сервісу та частково при його експлуатації зробити його безкоштовним. Головне – зачепити користувача на гачок тим, що новий SaaS – це проста у використанні і ефективна розробка.

У хмарного сервісу зручно налаштувати електронну оплату. Це спрощує процес продажів і робить прозорим цей процес для покупця.

SaaS повинен мати властивості допомагати у навчанні користувача. Взагалі, SaaS розрахований на те, що користувач без сторонньої допомоги зможе запустити застосунок, розібратися в його роботі і отримає очікуваний результат. Тобто, користувач повинен сам себе навчати. Тому SaaS повинен мати навчальні можливості, а не бути розрахованим на експерта. Бажано, щоб SaaS був чимось на зразок квест-гри, щоб під час оволодіння застосунком користувач поступово знаходив його нові корисні можливості.

У застосунок повинні бути вбудовані функції обліку досвіду користувача – які моделі і якої складності він будує, який час займає їхня оптимізація тощо.

Якщо SaaS враховує цей досвід, то він може бути під нього оптимізований. Такий досвід – це надбання, якого немає у конкурентів.

Слід намагатись тримати SaaS подовше затребуваним. Для цього треба постійно удосконалювати застосунок, нарощувати його функціональність, інтегрувати з іншими SaaS, залучати нові гілки у дереві його обходу.

Слід організувати службу зворотніх зв'язків з користувачами. Через відгуки, питання споживачів та відповіді на них можна організувати співтовариство любителів нового застосунку SaaS. Це дасть енергійний поштовх поширенню SaaS по мережі.

Слід організувати базу знань застосунку, яка виглядає як сайт вікіпедії. Це означає, що знання користувачі за їх бажанням можуть об'єднуватися в базу знань спільноти любителів застосунку SaaS.

Важливим аспектом SaaS є вбудовані інструменти підрахунку, наприклад, трафіку. Для вдалого маркетингу і поліпшення SaaS тут також потрібні різноманітні метрики: скільки кліків зробив користувач, які користувачі і як використовували нові можливості застосунку, які звичайні шляхи їх навігації, статистика трафіку сайту і т.п.

Інструментальна мова програмування хмарних застосунків

1. *Мова програмування Java*

Java – це об'єктно-орієнтована мова пограмування, яка розроблена і поширюється компанією Sun Microsystems як основний компонент платформи Java. Синтаксис мови багато в чому походить від C та C++. У офіційній реалізації платформи Java програми компілюються у байт-код, який при виконанні інтерпретується віртуальною Java-машиною.

Java пропонує своїм користувачам виключну кросплатформеність. Програма, написана на Java може працювати на будь-якій машині, куди можна встановити Java-машину. Це відкриває багатьом компаніям додаткові можливості, завдяки переходу на відкрите програмне забезпечення, де є підтримка платформи Java у повній мірі.

Основні переваги мови Java:

- Кросплатформеність. Програма може працювати на різних платформах без змін у коді або перекомпіляції.
- Звичний синтаксис. Мова значно запозичила синтаксис з C та C++. Знайомий синтаксис робить розробку проектів на Java більш інтенсивною.
- Java дозволяє викликати функції з сторонніх бібліотек. Якщо не вистачає функціонала, що пропонують стандартні бібліотеки – можна скористатися готовими вільними бібліотеками, які містять необхідний функціонал. Це значно розширює можливості інструменту програмування та дозволяє економити час на вже готових рішеннях. Завдяки цьому розробка проектів на Java більш інтенсивна.
- Платформа сервлетів. Сервлет – це стандартизований API, що дозволяє, використовуючи платформу Java, створювати динамічний контент до веб-сервера. Це дозволяє розвантажити клієнтські робочі

термінали та перекласти обчислення на сервер, який має значно більші обчислювальні потужності. Таким чином користувач може отримувати результат обчислень швидше, виконувати на своїй машині більше завдань без необхідності збільшення власної обчислювальної потужності. В свою чергу, потужний сервер стає дешевшим прямо пропорційно кількості клієнтів, які він обслуговує.

2 Додаткові засоби

XML - це запропонований консорціумом World Wide Web (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. Він є спрощеною підмножиною мови розмітки SGML. XML документ складається із текстових знаків, і придатний до читання людиною.

Стандарт XML (англ. *Recommendation*, перше видання від 10 лютого 1998, останнє) визначає набір базових лексичних та синтаксичних правил для побудови мови опису інформації шляхом застосування простих *тегів*. Цей формат достатньо гнучкий для того, щоб бути придатним для застосування в різних галузях. Іншими словами, запропонований стандарт визначає метамову, на основі якої шляхом запровадження обмежень на структуру та зміст документів визначаються специфічні, предметно-орієнтовані мови розмітки даних. Ці обмеження описуються мовами схем (англ. *Schema*), таких як DTD, RELAX NG або XML Schema. Прикладами мов, основаних на XML є: RSS, MathML, GraphML, XHTML, Scalable Vector Graphics, і також XML Schema.[18]

Висновки до лекції

Хмарне середовище відрізняється тим, що воно організоване як рівень програмної абстракції, який є вищим за рівень операційних систем. Через це, з одного боку, досягається проста організація обчислювального процесу на будь-якій обчислювальній платформі, яка є віддаленою, а з іншого боку, через накладні витрати на реалізацію ще одного поміжного рівня абстракції на сьогодні хмарні обчислення не доцільно застосовувати для високопродуктивних обчислень та обробки великих об’ємів даних.

Найбільш поширеною схемою застосування хмарних обчислень є і буде клієнт-серверна схема з клієнтом, реалізованим у веб-браузері та сервері, який запущено у певному вузлі Грід-системи чи комп’ютерного кластеру, який забезпечує подачу html-сторінок до веб-браузеру. Тому хмарні обчислення можна успішно розвивати за допомогою існуючих засобів розробки веб-сторінок та сайтів.

Microsoft® .NET Services

.NET Services надає основні стандартні блоки, які знадобляться при побудові додатків в хмарі і працюючих з хмарою для Azure ™ Services Platform. Сервіси, зібрані під ім'ям .NET Services, забезпечують інфраструктуру хмари, яка, в кінцевому рахунку, спрощує побудову працюючих в хмарі додатків. Сьогодні .NET Services забезпечують основну функціональність, пов'язану з можливостями підключення додатків, управління доступом і взаємодії за

допомогою повідомлень на базі робочого процесу. Під іменем .NET Services об'єднані наступні основні блоки сервісів:

• *Microsoft® .NET Service Bus* - надає мережеву інфраструктуру для з'єднання додатків через Інтернет з використанням різноманітних шаблонів обміну повідомленнями способом, забезпечуючим можливість проходження міжмережевих екранів і NAT-пристроїв без порушення безпеки, яка надається цими пристроями.

• Microsoft® .NET Access Control Service - забезпечує управління доступом в хмарі на підставі тверджень. Він включає механізм перетворення тверджень, який об'єднується з постачальниками посвідчень, такими, наприклад, як Active Directory і Windows Live ID (WLID).

• Microsoft® .NET Workflow Services - надає інфраструктуру для розміщення та управління робочими процесами (WF), приділяючи основну увагу взаємодії через повідомлення за допомогою .NET Service Bus. Поставляється з новими діями WF і інструментами для розміщення та керування екземплярами робочого процесу.

Дані нові сервіси можна розглядати як .NET-інфраструктуру сервісів для хмари. Всі вони доступні через відкриті протоколи і стандарти, включаючи REST, SOAP, Atom / AtomPub і WS. Це означає, що розробники на будь-якій платформі можуть інтегруватися з цими сервісами. Однак, в спробі зробити все максимально звичним для .NET-розробників, Microsoft також надає .NET Services SDK, який забезпечує хороші умови для .NET-розробника і приховує багато складних моментів роботи з сервісами. .NET Services SDK дозволяє розробникам використовувати наявний досвід .NET- розробки, зокрема в областях WCF і WF, через застосування нових розширень інфраструктури SDK (наприклад, нових прив'язок, каналів і дії). SDK також включає підтримку інструментів Visual Studio для інтеграції з порталом Azure ™ Services Portal. Крім .NET Services SDK, сьогодні партнери Microsoft пропонують Java і Ruby SDK.

Щоб почати роботу з .NET Services, перейдіть на портал Azure ™ Services Platform за адресою http://azure.com і клацніть посилання «*Try It Now*». Ви перейдете на сторінку «*Register for Azure Services*» (Реєстрація для сервісів Azure), представлену на рис.7.2.1. На цій сторінці даються важливі посилання для скачування різних SDK, доступу до додаткових ресурсів і переходу на сайт Microsoft Connect, де можна зареєструватися для отримання коду запрошення.

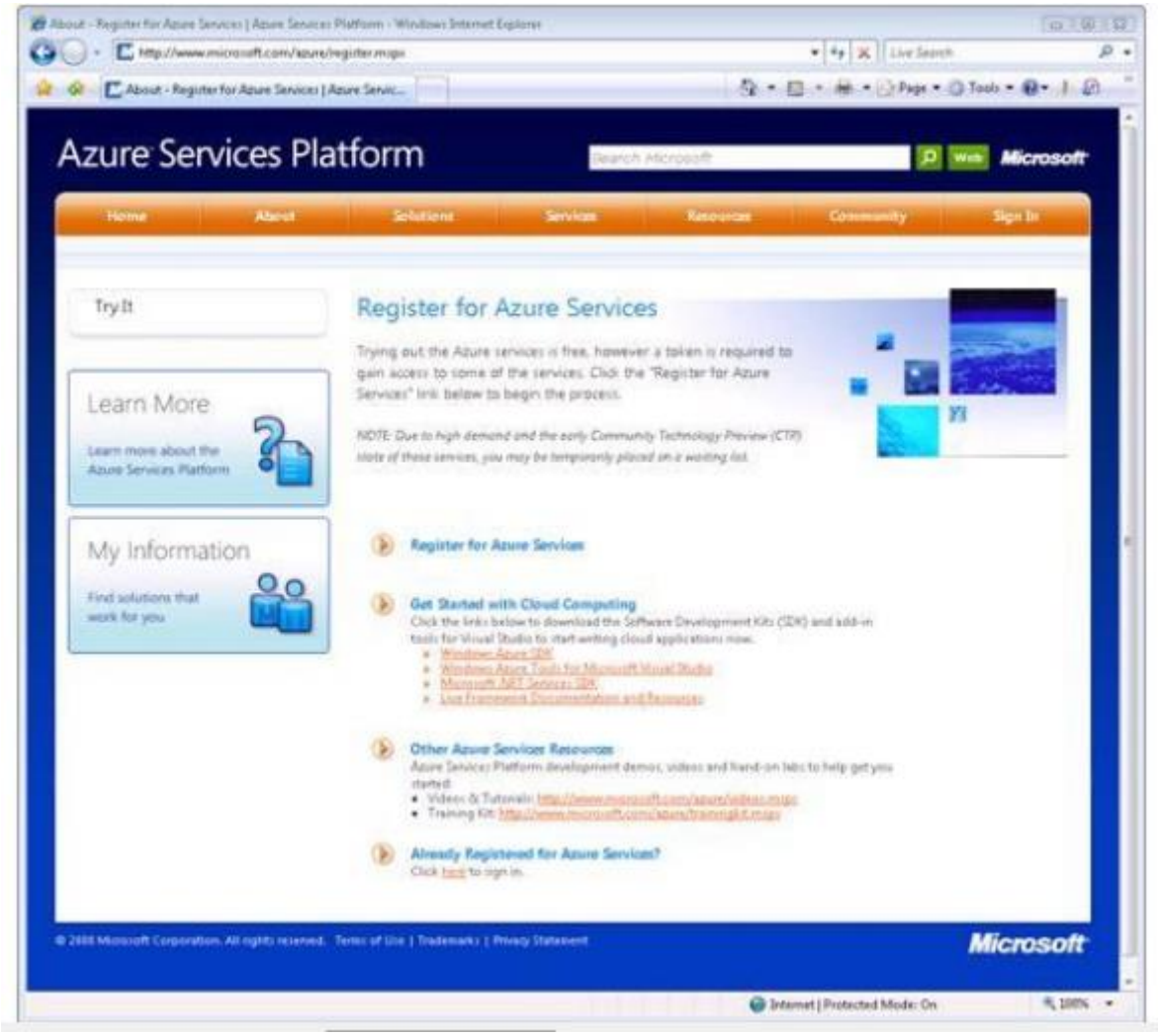


Рис.7.2.1 Портал Azure ™ Services Platform

Далі буде потрібно завантажити .NET Services SDK. Зверніть увагу, що є кілька SDK: один - спеціально призначений для розробки Windows® Azure ™; інший - для розробки .NET Services; і інші - для SQL Data Services і Live Framework. Для відтворення прикладів, пропонованих в даній серії документів, знадобиться завантажити і встановити .NET Services SDK.

Завантаживши .NET Services SDK, просто запустіть програму установки, як показано на рис.7.2.2 Тим самим Вам будуть доступні нові .NET-збірки, які разом з деякими надбудовами Visual Studio допоможуть розпочати використання різних функцій .NET Services. Приступаючи до роботи з .NET Services, обов'язково ознайомтеся з іншими ресурсами, доступними з цієї сторінки (демонстрації, відео, практичні лабораторні і т.д.), мета яких - зробити процес навчання більш насиченим і різноманітним. Завантажити SDK можна, не створюючи облікового запису, але, щоб використовувати сервіси, необхідно зареєструватися.

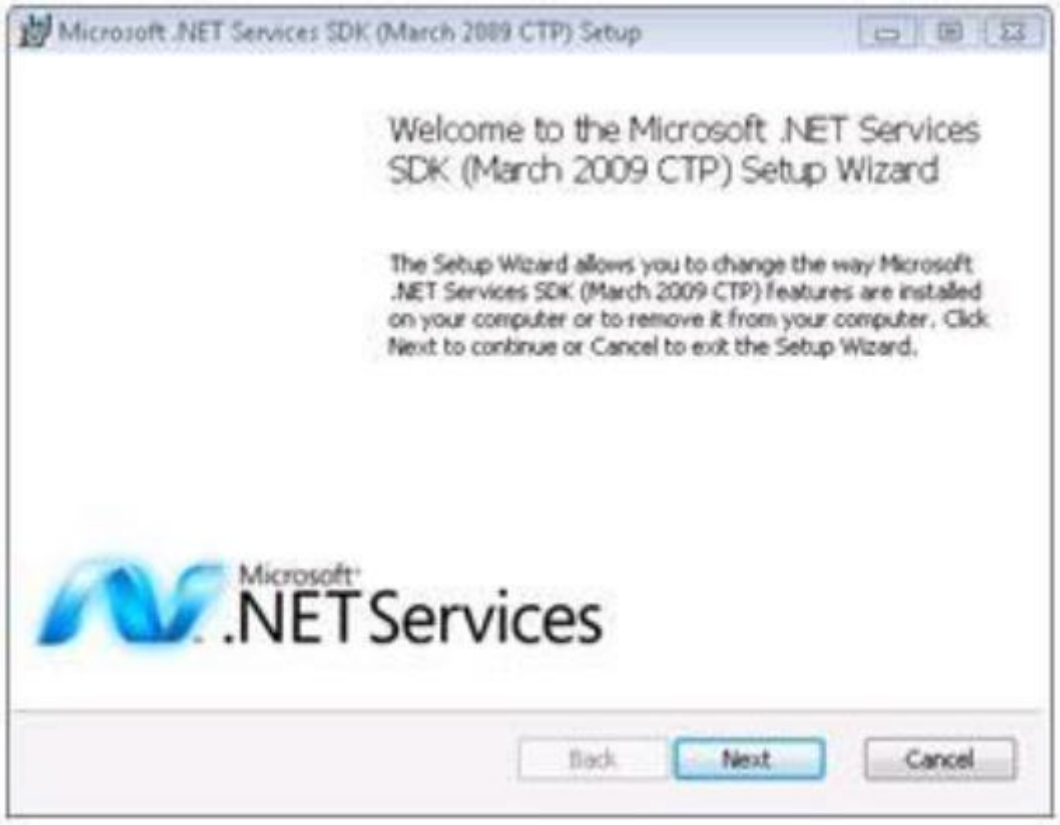


рис.7.2.2.Запуск установки .NET Services SDK

Щоб зареєструватися на отримання облікового запису Azure Services, клацніть посилання «*Register for Services*» (Реєстрація для сервісів). Вам буде запропоновано зареєструватися, використовуючи Windows Live ID (WLID). Після цього Ви перейдете на сайт Microsoft Connect, де потрібно заповнити реєстраційну форму Azure Services CTP. Після успішної реєстрації на Azure Services CTP, на екрані з'явиться сторінка, представлена на рис.7.2.3.

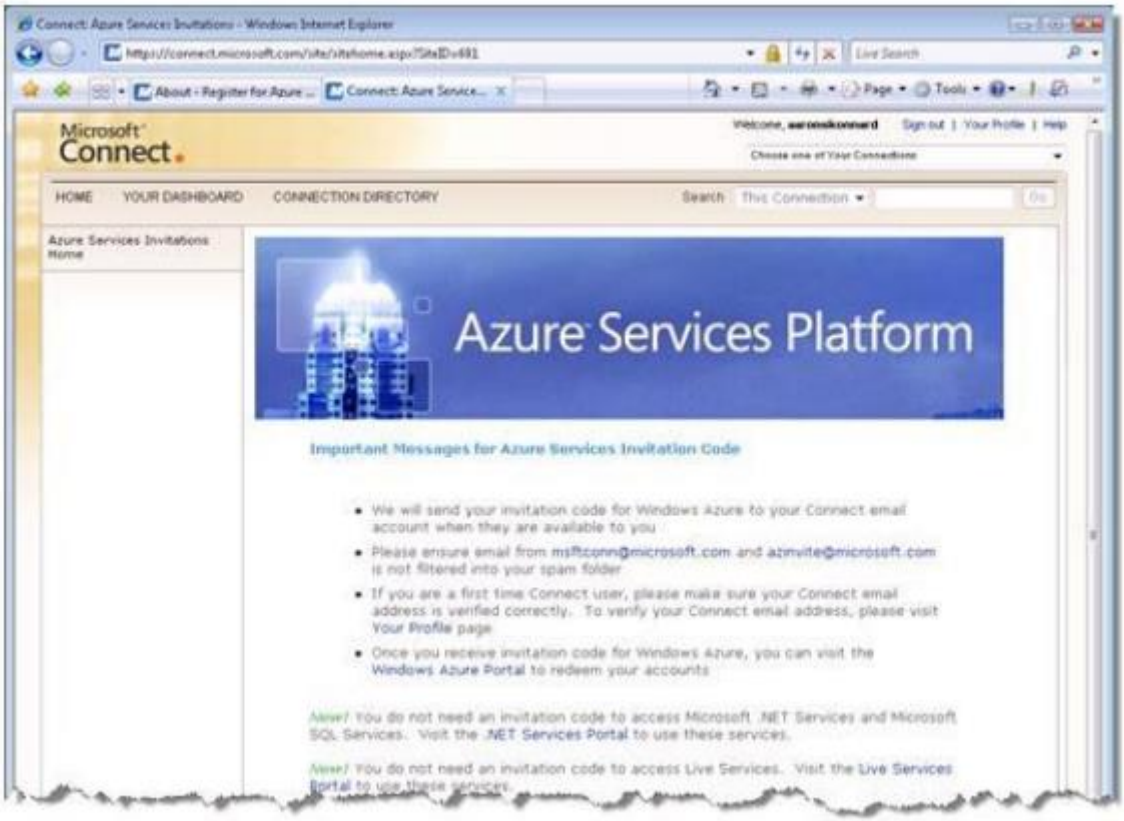


Рис.7.2.3 Реєстрація для Azure Services Platform на сайті Microsoft Connect

Тепер можна повернутися на сторінку входу .NET Services. Цю сторінку можна побачити на рис.7.2.4.

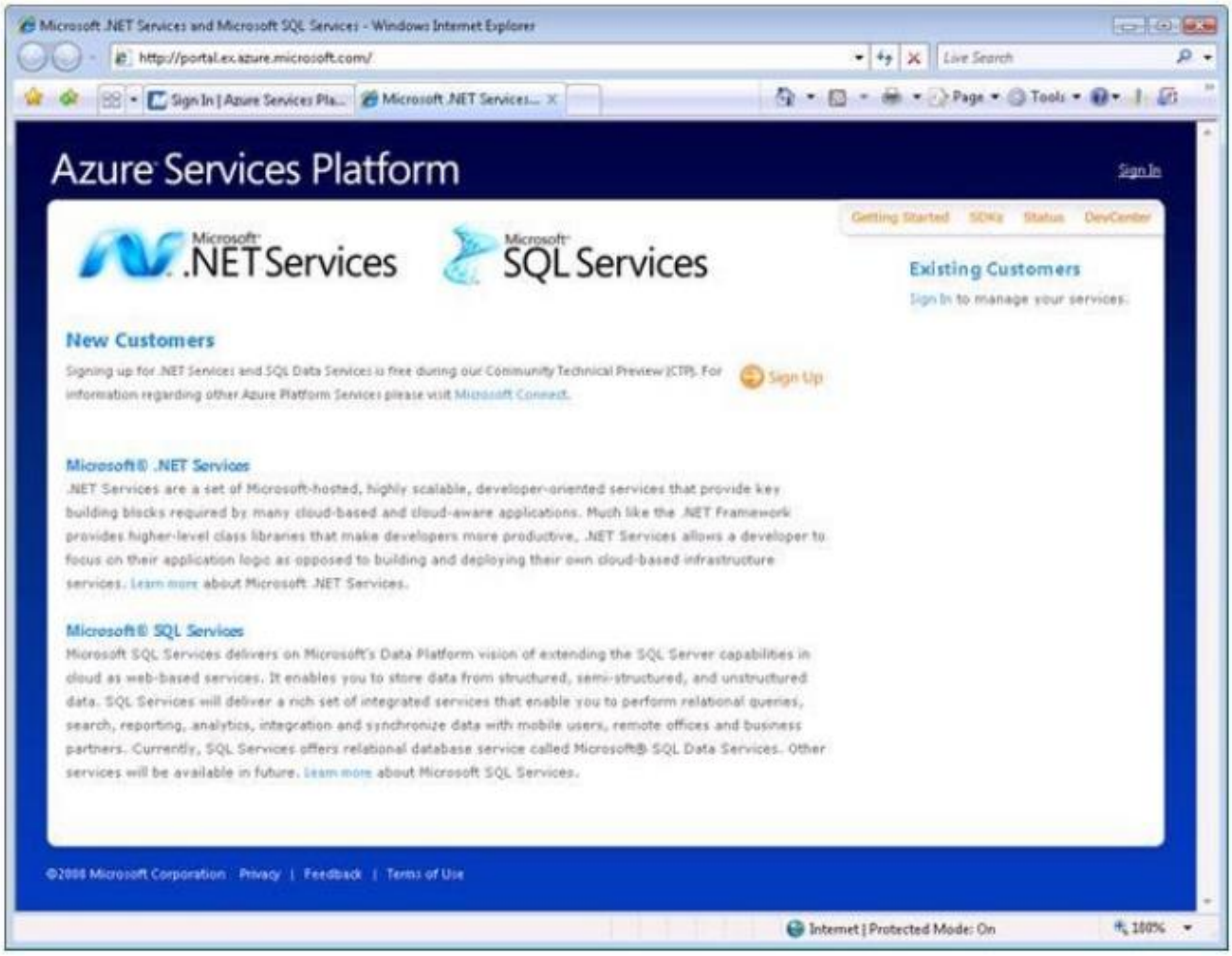


рис.7.2.4 Сторінка входу .NET Services

Тепер, клацнувши «*Sign Up*» (Ввійти), Ви отримуєте можливість створювати рішення .NET Services. Примітка: в CTP-версії для створення рішення .NET Services не потрібно код запрошення. Для створення рішення необхідно просто ввести унікальне ім'я рішення, прийняти умови використання і натиснути «*Create Solution*» (Створити рішення). Після цього нове рішення буде підготовлено і асоційоване з вашим WLID. Тепер в будь-який момент, зареєструвавшись на порталі Azure ™ Services Platform, Ви маєте можливість керувати рішеннями, асоційованими з вашим WLID. Ім'я рішення повинно бути не менше 6 символів довжиною і глобально унікальним серед всіх користувачів .NET-сервісів. Можливо, доведеться проявити кмітливість, щоб придумати таке ім'я для рішення, яке не використовується ніким іншим. Після того, як нове рішення створено, на екран виводиться сторінка (рис.7.2.5), яка пропонує пароль рішення, який бажано зберегти для використання в майбутньому. Ім'я рішення і пароль виступають в ролі облікових даних для доступу до різних сервісів .NET Services.

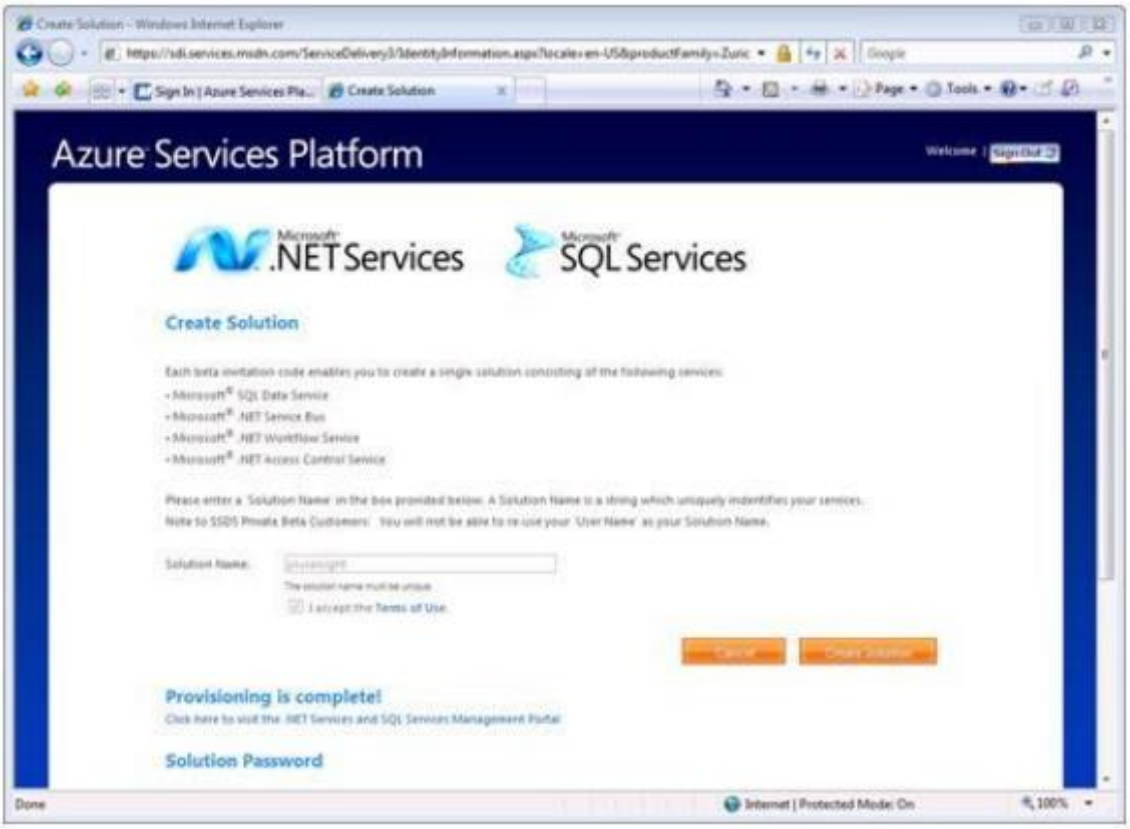


Рис.7.2.5 Завершення процесу підготовки рішення

Після успішного створення рішення можна приступати до роботи з ним на порталі Azure ™ Services Platform. Зареєструвавшись під власним WLID, на сторінці порталу справа Ви побачите меню «*My Solutions*» (Мої рішення) (рис.7.2.6). Для роботи з конкретним рішенням необхідно просто вибрати його в меню «*My Solutions*», після чого Вам буде представлена сторінка, показана на рис.7.2.7.

По суті, рішення - це контейнер верхнього рівня для організації різних ресурсів .NET Services. Наприклад, в ньому розміщуються кінцеві точки .NET Service Bus, типи і екземпляри робочих процесів .NET Workflow Service, а також ваші посвідчення .NET Access Control Service і правила перетворення тверджень. Але одним з найважливіших аспектів, яким Ви захочете керувати після створення власного рішення, є облікові дані рішення. Саме тому ім'я рішення повинно бути унікальним серед всіх користувачів.

Рис.7.2.6. Керування своїми рішеннями за допомогою меню «My Solutions»

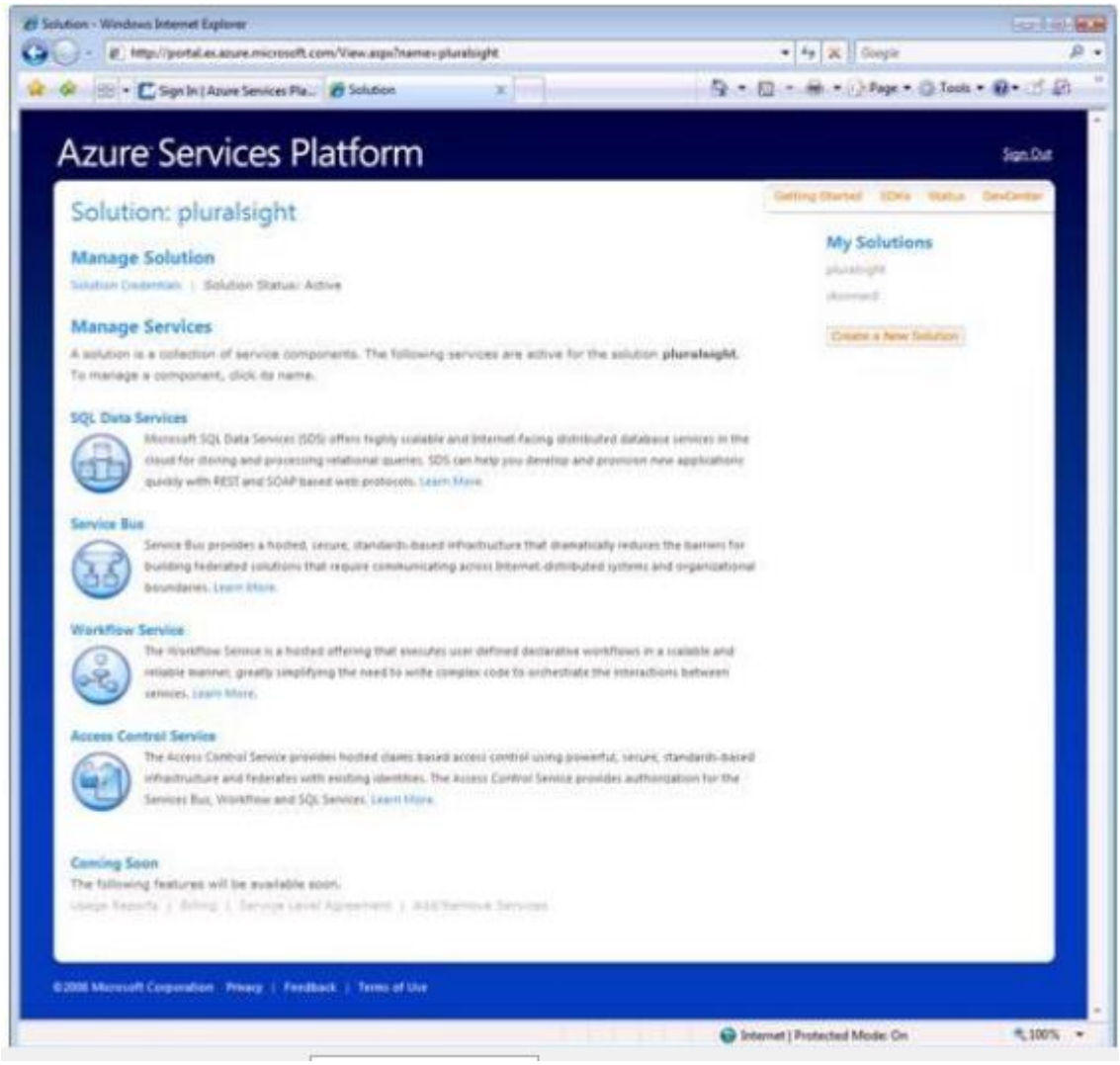


Рис.7.2.7 Керування окремим рішенням

Пароль рішення, наданий в процесі підготовки, можна змінити на сторінці «Credential Management» (Керування обліковими даними) (просто клацніть посилання «*Solution Credentials*» (Облікові дані рішення), яку можна

бачити на рис.7.2.8. З цієї сторінки можна також конфігурувати інформаційні картки Windows CardSpace, асоційовані з даним рішенням, і також будь-які сертифікати, які необхідно асоціювати з рішенням (рис.7.2.8).

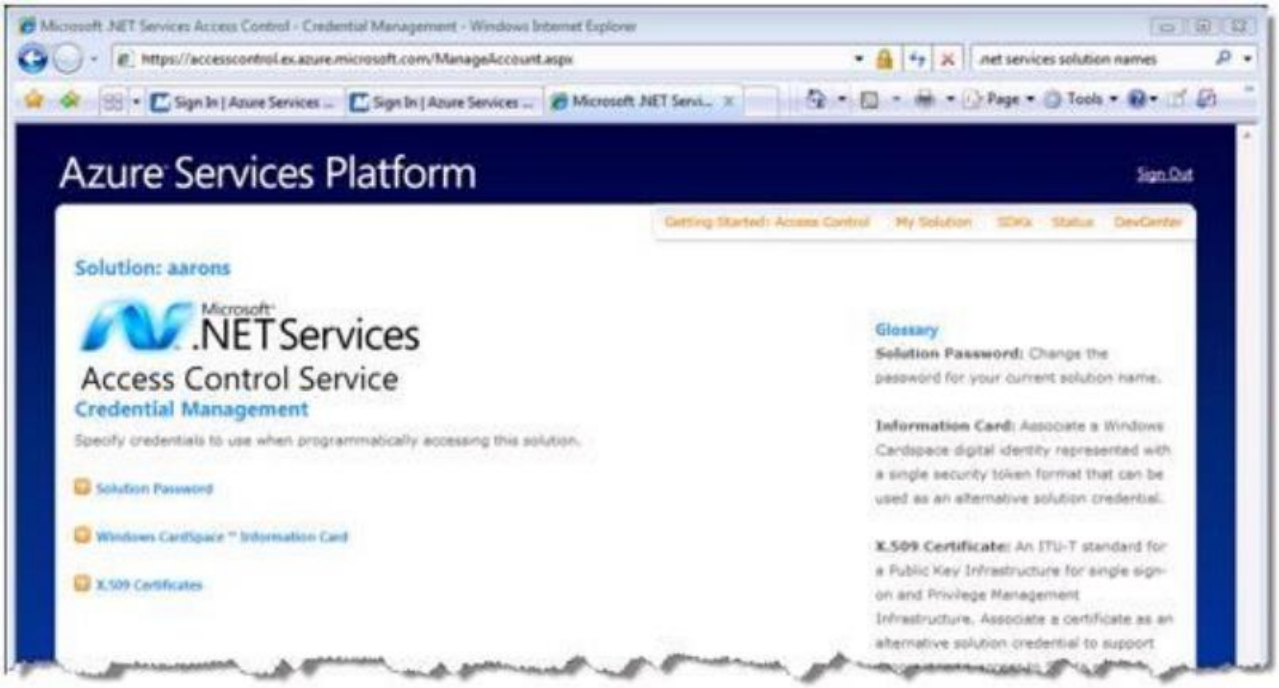


Рис.7.2.8 Керування обліковими даними свого рішення

Для Windows CardSpace і сертифікатів Вам запропонують вибрати необхідну картку / сертифікат, після чого відповідна інформація буде передана в обліковий запис Вашого рішення. З цього моменту в поєднанні зі своїм обліковим записом Ви можете використовувати облікові дані зазначеної картки / сертифікату. Найпоширенішою вимогою в розподілених додатках з високим рівнем масштабованості є можливість підключення додатків. Зазвичай інтеграція додатків - одна з найдорожчих і морочливих областей IT. Сьогодні для цих завдань багато організацій використовують рішення Enterprise Service Bus.

Enterprise Service Bus (сервісна шина підприємства, ESB) - підхід до побудови розподілених корпоративних інформаційних систем. Зазвичай включає в себе проміжне ПЗ, яке забезпечує взаємозв'язок між різними додатками з використанням різних протоколів взаємодії. Архітектура ESB полягає у взаємодії всіх додатків через єдину точку, яка, при необхідності, забезпечує транзакції, перетворення даних, збереження звернень. Даний підхід забезпечує більшу гнучкість, простоту масштабування і перенесення: при заміні одного додатку, підключеного до шини, немає необхідності переналпштовувати інші.

Одним зі стандартів взаємодії є веб-сервіси. У популярних реалізаціях ESB додаються шлюзи для обміну даними з корпоративним ПЗ. З використанням ESB може бути реалізована сервісно-орієнтована архітектура. Існує деяка розбіжність, що саме вважати ESB – архітектуру, чи програмне забезпечення. Обидві точки зору мають право на існування.

.NET Service Bus є основною частиною пропозиції .NET Services. Її основне завдання - зробити шаблон ESB реальністю в Інтернеті в рамках платформи Azure ™ Services Platform. Надані .NET Service Bus архітектурні характеристики багато в чому аналогічні запропонованим типовими рішеннями ESB, включаючи ідентифікацію та керування доступом, присвоювання імен, реєстр сервісу і загальне середовище обміну повідомленнями. Основна відмінність в області застосування. У випадку з .NET Service Bus компоненти повинні розроблятися для роботи в хмарі, в глобальній області Інтернету, із забезпеченням високого рівня масштабованості і інтегрованості. Саме тому в минулому цей пропонований сервіс називався Microsoft Internet Service Bus (рис.7.2.9).

Internet Service Bus дозволила б інтегрувати Ваш локальний ESB-продукт з вашими власними хмарними сервісами, з різними сторонніми сервісами, наданими Microsoft, або іншими виробниками (такими, як пропонуються в рамках платформи Azure ™ Service Platform) і з різними настільними, RIA3 і веб- додатками, які можуть виконуватися на допоміжних майданчиках поза брандмауера корпорації. Щоб це стало реальністю, реалізація повинна забезпечувати інтегровані рішення, засновані на відкритих Інтернет-стандартах, і насичене середовище обміну повідомленнями з можливістю двостороннього зв'язку в Інтернет.

1. Сервісна шина підприємства.
2. Ця термінологія застосовувалася в документації BizTalk Services, але більш не є офіційним найменуванням, використовуваним Microsoft.
3. RIA = Rich Internet Application (Насичений Інтернет- додаток).

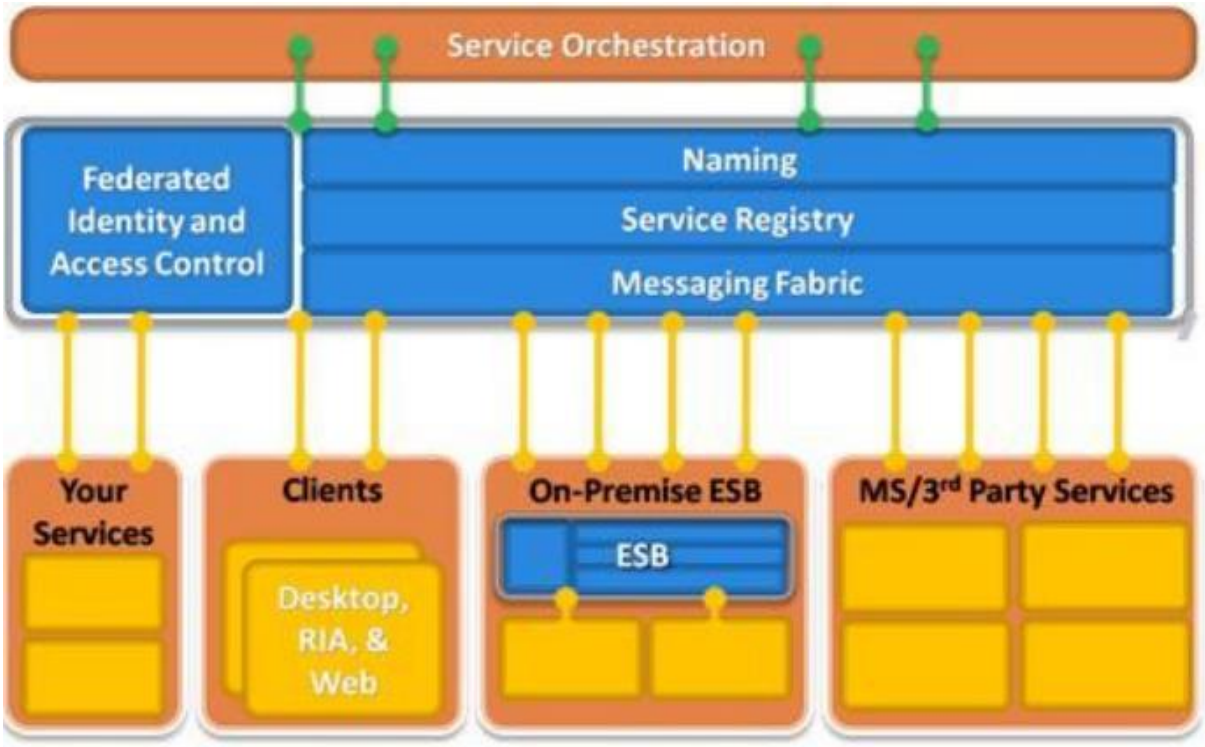


Рис.7.2.9 Сервісна шина Інтернет

Реалізація двостороннього зв'язку в Інтернеті не таке вже тривіальне завдання із-за деяких реалій організації сучасних мереж. Переважно бар'єри в

мережі створюють міжмережеві екрани і пристрої, які працюють по протоколу NAT, і які ускладнюють обмін інформацією з вузлами, розташованими за ними. Уявіть ситуацію: торговий агент використовує Ваш додаток по бездротовій мережі в випадковій готелі в деякій точці земної кулі. Як при такому сценарії визначити його місце розташування і ініціювати зв'язок?

Часто компанії вирішують ці проблеми зв'язку, відкриваючи вхідні порти міжмережевих екранів (що завдає чимало клопоту системним адміністраторам), або використовуючи різні обхідні прийоми, такі як динамічна DNS, зіставлення портів NAT , або технологію UpnP. Всі ці методи нестійкі, важко керовані і сприйнятливі до загроз безпеки. Кількість додатків, для яких потрібно такий тип двостороннього зв'язку, стає дедалі більше. .NET Service Bus покликана задовольнити цю потребу.

Network address translation - Перетворення мережевих адрес.

Domain Name System - Служба доменних імен.

Universal Plug and Play – Універсальне автоматичне налаштування мережевих пристроїв.

Рішення ідентифікації, реалізацією якої Microsoft займається останні кілька років, засноване на концепції тверджень. Модель ідентифікації на базі тверджень дозволяє виносити загальні функції аутентифікації і авторизації з додатків і здійснювати їх централізовано в зовнішніх сервісах, написаних і обслуговуюємих експертами безпеки та ідентифікації, що вигідно всім, хто бере участь в цьому процесі.

Microsoft® .NET Access Control Service - це сервіс в хмарі, який виконує саме цю функцію. Замість того, щоб створювати власну базу даних користувацьких облікових записів і ролей, можна надати можливість .NET Access Control Service керувати аутентифікацією і авторизацією Ваших користувачів. .NET Access Control Service використовує існуючі сховища облікових записів користувачів, такі як Windows Live ID і Active Directory, а також будь-які інші сховища, які підтримують стандартні протоколи інтегрування. Таким чином, використання єдиної реєстрації для доступу до всіх програм стає цілком природним. Також це забезпечує централізацію логіки аутентифікації і керування доступом, що спрощує Ваші додатки.

В підтримуючих твердження додатках користувач представляє своє посвідчення як набір тверджень. Одним твердженням може бути ім'я користувача; іншим - його адреса електронної пошти. Ці твердження надаються організацією, яка видає посвідчення, яка знає, як аутентифікувати користувача і де знайти його атрибути. Клієнтський додаток, в ролі якого може виступати браузер, або насичений клієнт, безпосередньо отримує твердження від цієї організації і передає їх в Ваш додаток (рис.7.2.10).

рис.7.2.10: Використання ідентифікації на базі тверджень для веб-сервісів

У підсумку додаток отримує всі відомості, необхідні для ідентифікації користувача, у вигляді набору тверджень. Ці твердження підписуються, що забезпечує криптографічне підтвердження їх походження. Модель ідентифікації на базі тверджень спрощує реалізацію єдиної реєстрації, при цьому додаток більше не відповідає ні за один з перерахованих нижче аспектів безпеки:

• Аутентифікація користувачів
• Зберігання облікових записів користувачів і паролів
• Звернення до каталогів підприємства в пошуках даних посвідчення користувача
• Інтеграція з системами посвідчень інших платформ або компаній

Використовуючи таку модель, додаток може приймати рішення про ідентифікацію на підставі наданих користувачем тверджень. І діапазон таких рішень великий: від простої персоналізації додатку по імені користувача до авторизації користувача для доступу до особливо важливих функцій і ресурсів додатку.

.NET Access Control Service реалізовує ідентифікацію на базі тверджень в рамках платформи Azure ™ Services Platform. Система адміністрування є важливою частиною .NET Access Control Service.

.NET Access Control Service надає портал адміністрування (рис.7.2.11) в рамках порталу Azure ™ Services Portal. Тут Ви виконуєте налаштування правил, які визначають схему випуску тверджень для різних користувачів.

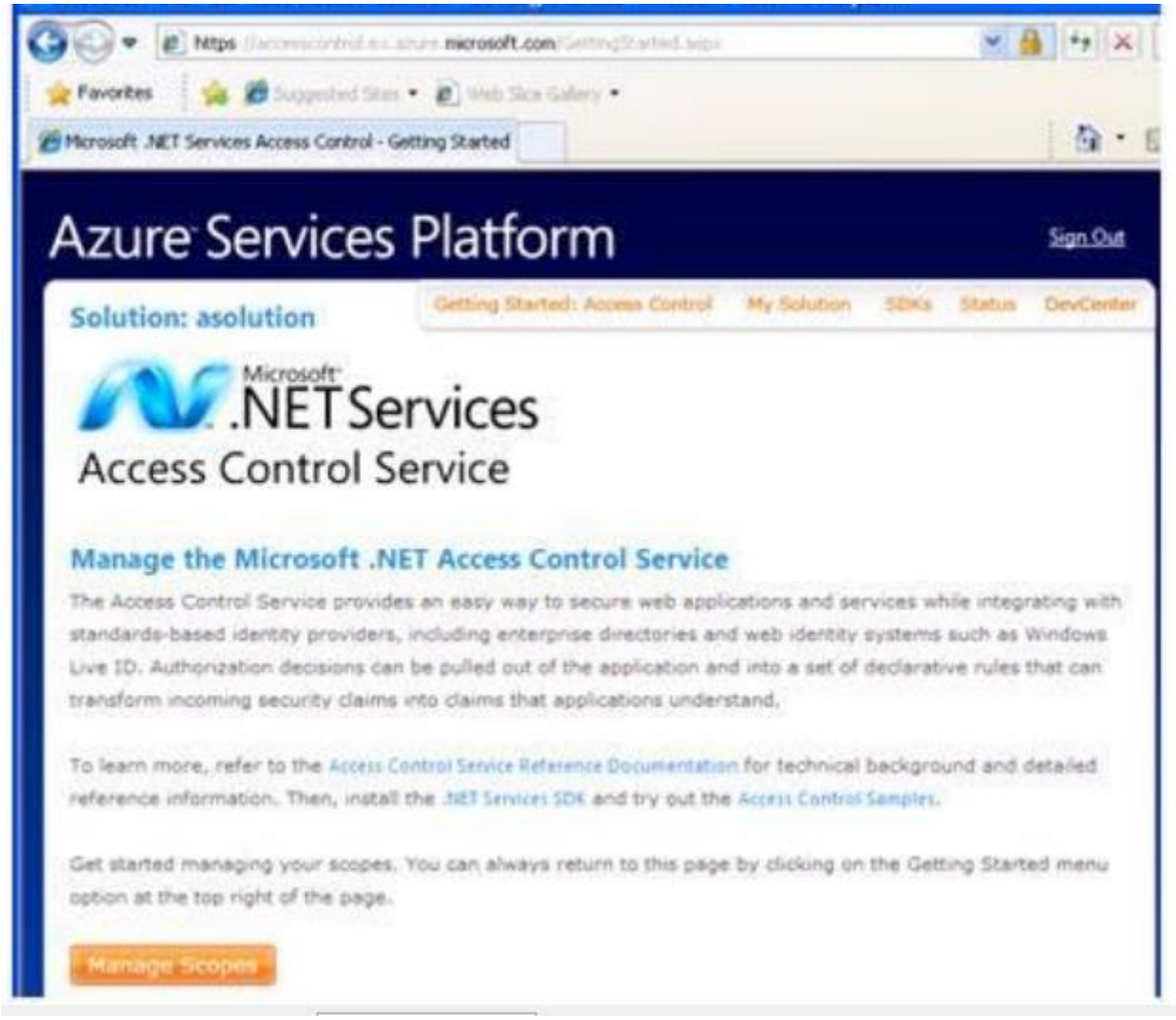


Рис.7.2.11. Портал ACS

Портал Access Control Service - хороший засіб для дослідження, вивчення і початку роботи з ACS. І для відносно простих додатків він може бути єдиним необхідним інструментом. Але для нетривіальних систем з сотнями або тисячами користувачів і, можливо, такою ж кількістю правил, використання порталу стає громіздким. У таких випадках програмний інтерфейс - більш кращий варіант, тому ACS також надає інтерфейс AtomPub для програмного адміністрування. AtomPub - це протокол RESTful, який стандартизує базові операції CRUD (Create, Retrieve, Update і Delete) для керування віддаленими ресурсами. Це відкриває абсолютно нові можливості. Сервіс .NET Access Control Service також включає кінцеві точки SOAP і REST для програмного адміністрування, а також ряд .NET-класів, які спрощують виклик цих кінцевих точок. Отже, якщо Вам не подобається портал, який надається ACS, або Ви бажаєте реалізувати налаштування, характерні для певної предметної області, можна створити власну консоль адміністрування.

Найбільшою проблемою в побудові великомасштабних розподілених додатків є прийняття рішення про моделювання складних схем взаємодії через обмін повідомленнями. Microsoft .NET Workflow Service дозволяє розробляти логіку взаємодії повідомлень за допомогою WF і забезпечує розміщене масштабіруєме середовище для виконання і керування екземплярами робочого

процесу WF в хмарі, звільняючи розробника від необхідності створення власного хоста для WF.

.NET Workflow Service є частиною Azure ™ Services Platform і інтегрується з сервісами .NET Service Bus і .NET Access Control Service для безпечного координування взаємодії за допомогою повідомлень. .NET Workflow Service також забезпечує інструменти керування для створення і керування типами та примірниками робочих потоків і API веб-сервісів для ситуацій, коли потрібно створити власні інструменти.

Оскільки керуюче середовище побудоване на платформі Windows® Azure ™, воно може масштабуватися на вимогу і в значній мірі при цьому організації, або розробнику не доводиться турбуватися про планування більшої кількості обладнання, або програмного забезпечення. Завдяки використанню середовища виконання WF екземпляри робочого потоку можуть виконуватися в пулі серверів і переміщуватися з одного сервера на інший в кожному епізоді виконання. Керуюче середовище включає сервіс зберігання, який використовує безпечні тиражовані сервіси Microsoft SQL Service для збереження стану виконуючихся робочих процесів і для забезпечення можливості відновлення. На період переходу до обробки даних в хмарі .NET Workflow Services надає спрощений підхід для керування складними взаємодіями .NET Service Bus в створюваних Вами складових рішеннях «в хмарі».

Побудова хоста для робочих процесів WF означає прийняття рішень про те, які можливості підтримуватиме середовище та як краще зробити її безпечною, масштабованою і стабільною. Сьогодні .NET Workflow Service побудований на .NET Framework 3.5 і діях і компонентах WF, які входять в дану версію інфраструктури. Однак для забезпечення найкращих умов Microsoft були додані декілька спеціальних дій і сервісів, які наклали деякі обмеження на робочі процеси, які виконуються в хмарі.

В хмарі не використовується сервіс зберігання *SqlWorkflowPersistenceProvider*, який отримав найбільшу популярність серед розробників, які використовують WF. Щоб використовувати операційне середовище Azure і забезпечити найкращі можливості масштабування і стабільності, в інфраструктурі хмари є спеціальний провайдер послуг зберігання, який реалізує збереження стану виконуючихся робочих процесів за допомогою можливостей зберігання Microsoft SQL Services. Крім усього іншого, для Інтернет-сервісу необхідна Інтернет-технологія зберігання та вилучення даних. Але оскільки механізм WF єдиний - як в хмарі, так і в Ваших локальних рішеннях, - застосування спеціального провайдера послуг зберігання прозоро для розробників робочих процесів. Все робиться так саме, як в будь-якому іншому середовищі WF.

При побудові робочих процесів для хмари розробники використовують звичні інструменти Visual Studio, включаючи той же дизайнер робочого процесу для створення XML-файлів робочих процесів і файлів правил. Потім ці XML-файли завантажуються на сервер в хмарі, де вони можуть використовуватися для створення екземплярів робочого процесу. .NET Services SDK включає шаблон проекту для створення *SequentialCloudWorkflow* (Послідовний робочий процес в хмарі), який є спеціальною версією

стандартного шаблону *SequentialWorkflow* (Послідовний робочий процес). Одним з обмежень поточної інфраструктури є те, що при визначенні робочих процесів, які будуть виконуватися в хмарі, можна використовувати тільки підмножину дій базової бібліотеки дій, а також комплект спеціальних дій, що надається як частина .NET Services SDK.

Набір дій вимагає, щоб робочі процеси були повністю декларативними і обмежуючими. Це запобігає введенню користувацького коду, тобто дозволяє гарантувати стабільність середовища. При побудові керуючого середовища для робочих процесів, написаних будь-якою кількістю розробників, розкиданих по всьому світу, такий рівень контролю є обов'язковим. Оскільки сьогодні для виконання WF необхідно повна довіра, ми не можемо забезпечити обмежений набір функціональності, просто виділивши користувальницький код в безпечне ізольоване програмне середовище на серверах. Слід зазначити, що з часом доступний сьогодні обмежений набір дій буде розширено для збільшення можливостей робочого процесу в хмарі. У міру виходу нових версій .NET Framework і .NET Workflow Service також буде підтримувати їх. Крім того, якщо знадобляться спеціальні етапи, можливості локальних робочих процесів можуть комбінуватися з робочими процесами в хмарі за допомогою .NET Service Bus.

При написанні робочих процесів в хмарі необхідно бути акуратним з використовуваними діями (дизайнер запропонує тільки допустимі дії). Дозволеними є деякі основні дії потоку керування WF, включаючи *IfElse* (Якщо ... то), *While* (Поки), *Sequence* (Послідовність), *Suspend* (Призупинити), *Terminate* (Завершити) і *FaultHandler* (Оброблювач збоїв). Крім базових дій потоку керування, в робочих процесах в хмарі можуть також використовуватися *CancellationHandlerActivity* і *FaultHandlersActivity* для моделювання обробки виключень і логіки скасування. Зверніть увагу, що ці дії зазвичай не додаються в модель безпосередньо; для цього використовується дизайнер складових дій, який вводить їх автоматично, коли уявлення переходить до цього дійства.

Оскільки основним завданням .NET Workflow Service є спрощення взаємодії повідомлень, ці дії, головним чином, стосуються відправки, отримання та обробки повідомлень. Відправляти / приймати повідомлення можна за допомогою традиційних HTTP-запитів, або через .NET Service Bus. Ці дії можна буде знайти на панелі інструментів Visual Studio при використанні шаблону проекту послідовного робочого процесу в хмарі.

Отже, платформа Azure ™ Services Platform представляє комплексну стратегію, розроблену Microsoft для полегшення розробникам завдань щодо реалізації можливостей обробки даних в хмарі. Microsoft® .NET Services - ключова складова цієї платформи, створена спеціально, щоб допомогти .NET-розробникам зробити перший крок. .NET Services пропонує орієнтовані на роботу в хмарі стандартні блоки і інфраструктуру для забезпечення можливості підключення додатків, керування доступом, розміщення та керування робочим процесом. Ці стандартні блоки стануть основними засобами організації роботи «з хмарою» для .NET-розробників і в майбутньому. Більше інформації про

.NET Service Bus, .NET Access Control Service і .NET Workflow Service можна знайти в документах даної серії, присвячених кожній з цих тем окремо.

Приклади хмарних сервісів Microsoft і Google пропонується засвоїти в рамках самостійної роботи студента.

Питання для самоконтролю

1. Охарактеризуйте програмне забезпечення хмарних обчислень.
2. Перерахуйте особливості програмування в «хмарах».
3. Що таке консолідація і віртуалізація IT-інфраструктури?
4. Що таке хмарні обчислення і на що націлені хмарні обчислення?
5. Назвати обчислювальні ресурси «хмар».
6. Назвати етапи впровадження хмарних обчислень.
7. Що стало основою хмарних обчислень?
8. Що являє собою гіпервізорна технологія?
9. Охарактеризуйте архітектуру Hyper-v.
10. Охарактеризуйте особливості розробки застосунків для хмарного середовища.
11. Охарактеризуйте Microsoft System Center Virtual Machine Manager (SCVMM).
12. Що таке Плагіни та веб-платформи?
13. Перерахуйте види послуг, які надаються хмарними системами.
14. Що таке «Інфраструктура як сервіс» (Infrastructure as a Service , або IaaS)?
15. Що таке «Платформа як сервіс» (Plaatform as a Service , PaaS)?
16. Що таке «Програмне забезпечення як сервіс» (Software as a Service , SaaS)?
17 Що являє собою Microsoft® .NET Service Bus?
18 Що являє собою Microsoft® .NET Access Control Service?
19 Що являє собою Microsoft® .NET Workflow Services?
20 Що таке .NET Services SDK?
21 Що таке Microsoft® .NET Access Control Service?

Література

1.Windows Server 2008 R2 and Windows Server 2008. Режим доступу: www.microsoft.com/windowsserver2008/en/us/2008-dc. aspx.
2. Microsoft" Windows Server' 2012 (Rand Morimoto, Ph.D., MCITP, MVP, Michael Noel, MVP, MCITP, Guy Yardeni, MCITP, CISSP, MVP, Omar Droubi, MCSE, MCTS, Andrew Abbate, MCITP, Chris Amaris, MCITP, MCTS, CISSP, Technical Edit by Tyson Kopczynski, CISSP, GCIH)// SAMS, 800 East 96th Street, Indianapolis, Indiana 46240 USA.
3. J. Schopf; Ten Actions When Super Scheduling // document of Scheduling Working Group, Global Grid Forum. Режим доступу: http://www.ggf.0rg/d0cuments/GFD.4.pdf, July 2001.
4. Google App Engine. Режим доступу: http://code.google.com/appengine
5. Microsoft Azure. Режим доступу: http://www.microsoft.com/azure

6. Amazon Elastic Compute Cloud. Режим доступу: http://aws.amazon.com/ec2/
7. Apex: Salesforce on-demand programming language and framework. Режим доступу: http://developer. force.com/
8.Dean J. and Ghemawat S. MapReduce: Simplified data processing on large clusters // *Proceedings of the Sixth Symposium on Operating System Design and Implementation* (*OSDI*), San Francisco, CA, – 2004. –p. 137–150.
9.Hadoop. Режим доступу: http://hadoop.apache.org/
10.Amazon Simple Storage Service. Режим доступу: http://aws.amazon.com/s3/
11. Foster I. , Kesselman C.. Globus: A metacomputing infrastructure toolkit. *Int. J. Supercomput. Appl*., –1997. –V.11. – N2. –p.115–128.
12.D. Thain, T. Tannenbaum, Livny M.. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*. –2005. –V.17. N2–4. –p.323–356.
13.Jackson. K. L. An ontology for tactical cloud computing. Режим доступу: http://kevinljackson.blogspot.com/
14. Hoff C. Christofer hoff blog: Rational survivability. Режим доступу: http://rationalsecurity.typepad.
15 Cloud Computing and Software Services. Theory and Techniques. / S. A. Ahson, M.Ilyas – ed-s.–CRC Press Taylor & Francis Group. –2011. –442.
16. Adobe Flash runtimes. Benefits. Режим доступу: http://www.adobe.com/products/flashruntimes/benefits.html
17. York. J. SaaS Growth Strategy. A Customer Lifecycle Approach. –2013. –16 P. Режим доступу: http://chaotic-flow.com/media/saas-growth-strategy.pdf
18. Harold E. R. XML 1.1 Bible. 3rd Ed. Indianapolis: Wiley Publishing, Inc. - 2004. - 1022 P.