НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.855.5:004.855.6

До захисту допущено
В. о. завідувача кафедри ММСА
О.Л.Тимощук

«___» _____ 2020 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 122 Комп’ютерні науки
на тему: «Система оркестрації в розподілених обчислювальних мережах на основі навчання з підкріпленням»

Виконав:
студент ІІ курсу, групи КА-93мп
Міщук Андрій Романович _____

Керівник:
доцент кафедри ММСА,
к.т.н., доц. Дідковська М.В. _____

Рецензент:
доцент кафедри програмного забезпечення комп'ютерних систем
КПІ ім. Ігоря Сікорського, к.т.н., доц. Заболотня Т. М.
_____

Засвідчую, що в цій магістерській дисертації
немає запозичень із праць інших авторів
без відповідних посилань
Студент _____

Київ
2020

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність — 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___» _____ 2020 р.

ЗАВДАННЯ
на магістерську дисертацію студента Міщука Андрія Романовича

1. Тема дисертації: «Система оркестрації в розподілених обчислювальних мережах на основі навчання з підкріпленням», науковий керівник дисертації Дідковська Марина Віталіївна к.т.н., доцент, затверджені наказом по університету від 02 листопада 2020 р. № 3182-с.
2. Термін подання студентом дисертації: 14 грудня 2020 р.
3. Об'єкт дослідження: розподілені системи
4. Предмет дослідження: системи оркестрації на основі навчання з підкріпленням
5. Перелік завдань, які потрібно розробити:
1) дослідити наявні сисстеми оркестрації, принципи роботи та особливості
2) огляд підходів побудови систем на основі навчання з підкріпленням
3) обрати та обґрунтувати архітектуру системи оркестрації
4) тестування системи оркестрації
5) аналіз результатів роботи
6) розробити стартап-проект виведення на ринок результатів дослідження;
7) розробити концептуальні висновки за результатами наукового дослідження.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу
1. Рисунки схем роботи деяких системи
2. Таблиці порівняння результатів роботи
3. Рисунки та графіки на яких зображено результати роботи
4. Таблиці у розділі стартап-проекту
7. Дата видачі завдання: 1 вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації
1	Концептуальний вступ дисертації. Формулювання об’єкта, предмета, цілі, завдань, новизни, практичної значущості результатів	05.09.2020—13.09. 2020
2	Перший розділ. Огляд літературно-інформаційних джерел, формування нормативної бази. Характеристика об’єкта.	16.09.2020—27.09.2020
3	Другий розділ. Опис розробленої системи, середовища та технологій. Опис архітектури програмного рішення.	30.09.2020—18.10.2020
4	Третій розділ. Опис процесу розробки агента та аналіз результатів роботи.	21.10.2020—29.10.2020
5	Четвертий розділ. Стартап-проект	30.10.2020—17.11.2020
6	Концептуальні висновки. Перспективи розвитку отриманих рішень	22.11.2020—26.11.2020

Студент Міщук А.Р.
Науковий керівник дисертації Дідковська М.В.

РЕФЕРАТ

Магістерська дисертація: 70 с., 26 табл., 20 рис., 1 дод., 20 джерел.

НАВЧАННЯ З ПІДКРІПЛЕННЯМ, СИСТЕМИ ОРКЕСТРАЦІЇ, РОЗПОДІЛЕНІ СИСТЕМИ, ФУНКЦІЯ ВИНАГОРОДИ, АЛГОРИТМ SARSA

Проблема керування ресурсами є важливою для сучасних систем, оскільки, незважаючи на розвиток сучасних засобів керування розподіленими системами, контроль ресурсів досі значною мірою є відповідальністю адміністратора системи. Саме тому розробка автоматизованого оркестратора є сьогодні актуальною.

Об’єктом дослідження є розподілені системи.

Предметом дослідження є системи оркестрації на основі навчання з підкріпленням.

Метою цього дослідження є розв’язок проблеми оркестрації ресурсів з допомогою підходів навчання з підкріпленням. Тестування, аналіз та оформлення результатів роботи системи.

Для досягнення бажаного результату застосовано метод Expected SARSA, Adam та буфер відтворення досвіду.

Наукова новизна полягає у розв’язку проблеми оркестрації в розподілених системах підходами навчання з підкріпленням, оскільки сучасні рішення ґрунтуються на імперативному, цілком контрольованому людиною керуванні.

ABSTRACT

The master's thesis: 70 p., 26 tables, 20 fig., 1 add., 20 sources.

REINFORCEMENT LEARNING, ORCHESTRATORS, DISTRIBUTED SYSTEMS, REWARD FUNCTION, SARSA ALGORITHM

The problem of resource management is important for modern systems, because, despite the development of modern means of managing lean systems, resource control is still largely the responsibility of the system administrator. That is why the development of an automated orchestrator is relevant today.

The object of the research is distributed systems.

The subject of the research is systems based on orchestration systems.

The purpose of this study is to solve the problem of orchestration of resources using reinforced learning approaches. Testing, analysis and registration of system results.

To achieve the desired result, the Expected SARSA method, Adam and the experience playback buffer were used.

The scientific novelty lies in solving the problem of orchestration in distributed systems with reinforced learning approaches, as modern solutions are based on imperative, fully human-controlled management.

ЗМІСТ

ВСТУП

З розвитком напрямків ШІ постійно переосмислюються та змінюються інструменти, що були породжені комп'ютерними науками. Підходи та рішення, що вважались усталеним стандартом, зазнають кардинальних змін. Системи моніторингу та керування ресурсами не є винятком, адже досі не існує однозначного, гнучкого та універсального рішення. Наразі найбільш зрілим рішенням для розв'язку проблеми оркестрації ресурсів є системи, що ґрунтуються на обмеженнях та сценаріях, виставлених адміністратором системи. Такий підхід не є ідеальним, адже вимагає чіткого розуміння можливостей системи, прогнозу очікуваного навантаження та, найголовніше, контролю. Саме адміністратор мусить розуміти(вимірювати) можливості свого продукту та налаштовувати під нього оркестратор. Ця потреба і робить сучасні системи керування ресурсами негнучкими, що є критичним, адже постійна еволюція -- неодмінна характеристика будь якого продукту.

Перевагою даної роботи є те, що оркестратор здатен підлаштовуватись під динамічний продукт без втручання людини.

Мета роботи - розв'язок проблеми оркестрації ресурсів з допомогою навчання з підкріпленням.

Пояснювальна записка складається з чотирьох розділів. В першому розділі наводиться огляд сучасних систем керування ресурсами та відбувається детальний огляд підходів та можливостей навчання з підкріпленням в даному полі. Другий розділ описує обране рішення та його основні характеристики, метрики та параметри.

В третьому розділі міститься детальний огляд дослідження та аналіз прикладів зміни середовища та реакцій оркестратора. Четвертий розділ описує стартап-проект, створений на основі даного дослідження.

РОЗДІЛ 1. ОГЛЯД СУЧАСНИХ СИСТЕМ КЕРУВАННЯ РЕСУРСАМИ ТА ОГЛЯД ПІДХОДІВ НАВЧАННЯ З ПІДКРІПЛЕННЯМ

1.1 Системи оркестрації, призначення та затребуваність.

Оркестрація - автоматизоване конфігурування, керування та координація комп'ютерних систем, застосунків та сервісів. Оркестрація допомагає полегшити процес керування інфраструктурою та компонентами програмного продукту. Завдяки даному інструменту можна значно підвищити надійність продукту та знизити витрати на підтримку. Сучасні рішення автоматизують розгортання, масштабування, конфігурування мереж та управління сервісами. Оркестрація контейнерів може використовуватись в середовищах, де використовується контейнеризація. Це допомагає розгортати застосунки в різних середовищах без необхідності зміни або доробки програм.

Контейнери набули популярності завдяки наступним властивостям:

1) гнучке створення та розгортання застосунків, контейнери значно легші за традиційні VM образи;
2) постійна розробка, інтеграція та розгортання забезпечується надійною та частою збіркою контейнерів, розгортанням з швидким та легким відновленням при падіннях;
3) чітка межа між продуктом та інфраструктурою;
4) видимість не лише показників OS, а й безпосередньо застосунку;
5) цілісність розгортання не залежить від середовища;
6) чітке розмежування відповідальності кожного сервісу;
7) ізоляція ресурсів, передбачувана продуктивність;
8) цілковитий контроль над ресурсами(utilization).

Сьогодні найпоширенішими системами оркестрації є Docker Swarm та Kubernetes. На рис 1. зображено еволюцію розвитку віртуалізації та розгортання програм.

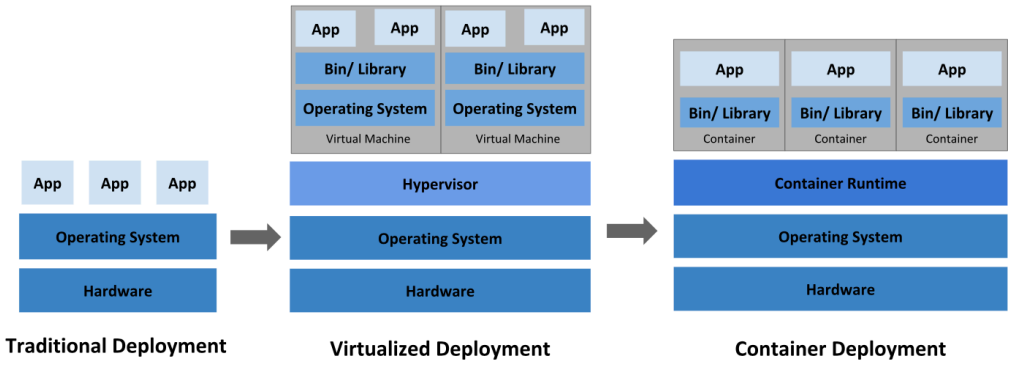

Рисунок 1 - Еволюція підходів розгортання програм

1.2 Рівні віртуалізації

П’ять рівнів реалізації віртуалізації:

1) Рівень архітектури набору інструкцій (ISA)
2) Рівень апаратної абстракції (HAL)
3) Рівень операційної системи
4) Бібліотечний рівень
5) Рівень застосування

Незважаючи на те, що існує п’ять рівнів віртуалізації, не обов’язково використовувати усі. Це залежить від того, над чим працює компанія та якому рівню віртуалізації вона віддає перевагу.

Компанії, як правило, використовували віртуальні машини для розробки та тестування кроссплатформенних додатків. З ростом хмарних додатків віртуалізація стала необхідною умовою для підприємств по всьому світу. В наступних розділах ми детальніше розглянемо дві такі технології - Docker Swarm та Kubernetes.

1.3 Docker Swarm

Docker Swarm - технологія, що дозволяє автоматизовано керувати групою фізичних чи віртуальних машин, які, в свою чергу, мають запущену програму-агент Docker. Цей інструмент дозволяє користувачу керувати кількома контейнерами, розгорнутими на різних машинах кластеру. Однією з найбільших переваг за використання Docker Swarm є високий рівень доступності керованих програм. Docker Swarm кластер складається з менеджера(manager node) та робочих машин(worker nodes). На рис. 2 зображено базову архітектуру Docker Swarm кластера.

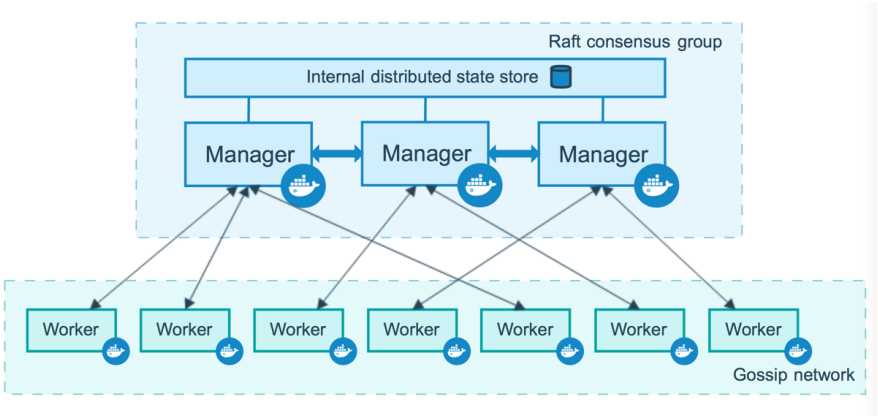

Рисунок 2 - Архітектура кластера Docker Swarm

Менеджер відповідальний за розподіл робочих машин(ресурсів) між сервісами.

Робочі машини отримують та виконують задачі, отримані від менеджера, статус виконання задач постійно відправляється менеджеру.

Сервіс в контексті Docker Swarm - це визначення задач, що необхідно виконати. Це є центральною точкою взаємодії з користувачем. Визначаючи сервіс, користувач вказує, з яких контейнерів він складатиметься, скільки реплік необхідно забезпечити, яку мережу та яке сховище використовувати. [19] На рис. 3 зображено приклад розгортання сервіса на кластері.

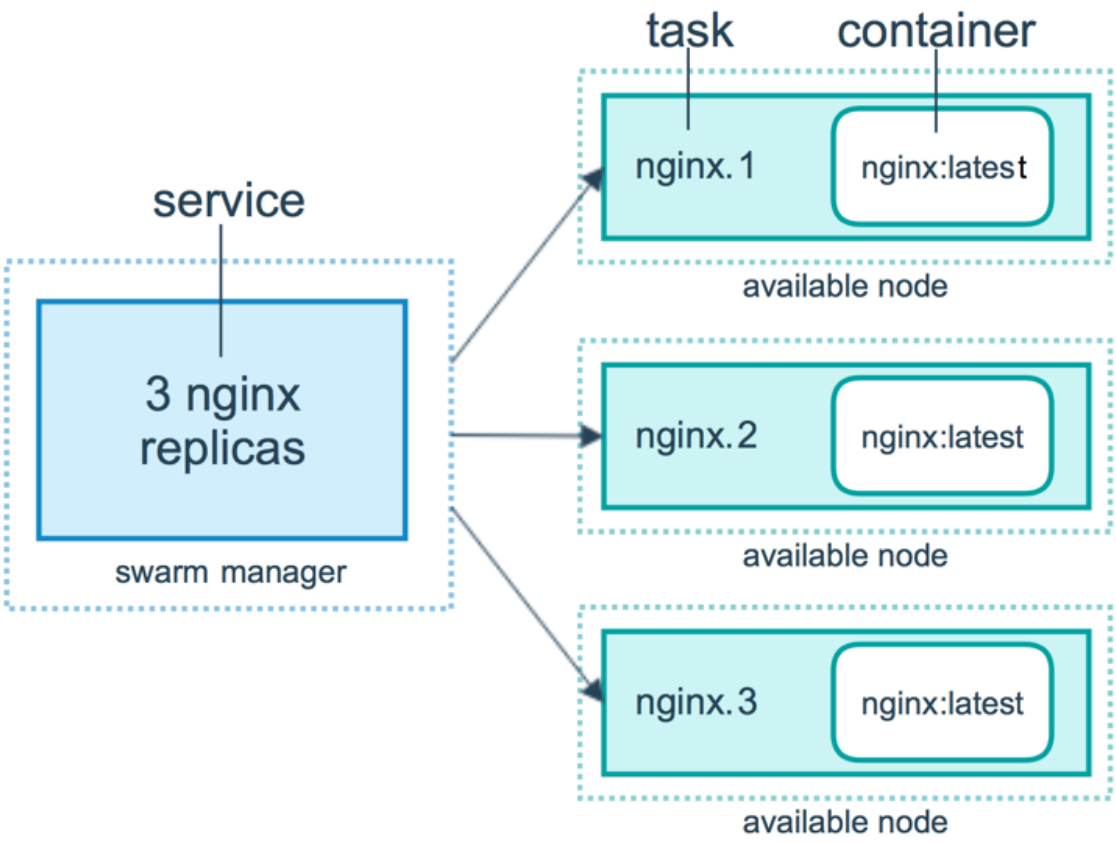

Рисунок 3 - Приклад сервіса nginx(синій) з рівнем реплікації 3

Задачі кластер менеджера:

1) підтримувати стан кластера;
2) запускати сервіси за розкладом(scheduling);
3) коректно направляти HTTP запити до відповідних нод;
4) підтримка високої доступності(high availibility);
5) відновлення після падіння менеджера без затримок.
6) конфігурації сервісів:
7) конфігурації сервісів:
8) відкриті порти та їх зв’язки(mappings);
9) обмеження CPU та RAM;
10) рівень реплікації;
11) політика поступового оновлення(rolling update policy).

1.4 Kubernetes

Kubernetes - розширювана, відкрита платформа для керування контейнеризованими навантаженнями та сервісами, що забезпечує декларативне конфігурування та автоматизацію. Це велика та стрімко зростаюча екосистема.[18]

Можливості Kubernetes:

1) балансування навантаження - Kubernetes вміє розподіляти трафік між контейнерами, цим самим підтримуючи стабільність сервісу;
2) керування сховищами - Kubernetes підтримує багато різних типів сховищ, в тому числі і хмарні;
3) автоматизоване відновлення після падінь;
4) розподіл ресурсів відповідно до лімітів;
5) самолікування - Kubernetes контролює стан контейнерів та перезапускає проблемні;
6) також Kubernetes дозволяє керувати паролями, ключами та токенами, що є необхідним для повноцінних сервісів.

Kubernetes кластер складається з робочих машин, кожна з яких може відповідати за підтримку роботи кількох под, що, в свою чергу, є складовими програм. На рис. 4 зображено компоненти кластера Kubernetes.

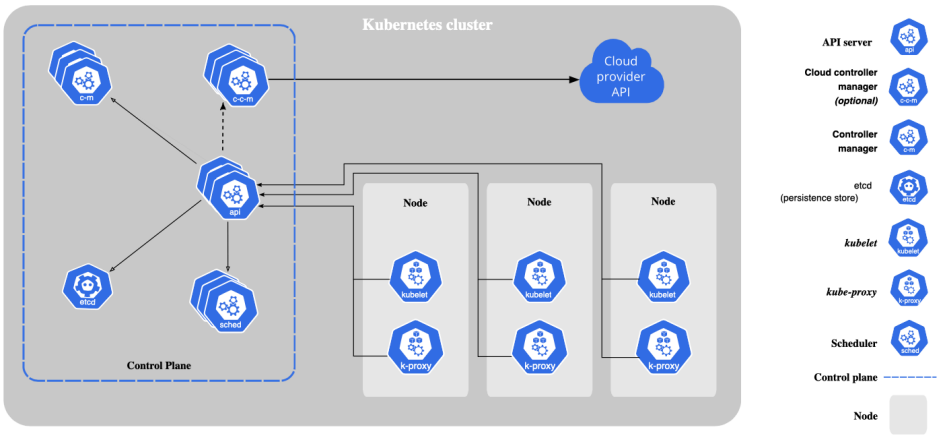

Рисунок 4 - Компоненти Kubernetes кластера

Важливою складовою є контрольна панель(Control Plane), вона включає в себе процеси керування робочими машинами, реплікацією, точками доступу, сервісними профілями та токенами.

1.5 Навчання з підкріпленням

Навчання з підкріпленням в значній мірі спирається на концепцію стан - як вхід для політики та функції значення. Неформально ми можемо розглядайте стан як сигнал, що передає агенту якесь відчуття середовища у певний час. Так, ми припустимо, що сигнал стану виробляється якоюсь номінальною системою попередньої обробки частина середовища агента. [12]

Основною метою навчання з підкріпленням є отримання алгоритму(агента), що визначатиме дії відповідно до стану середовища. Кінцевою метою є максимізація винагороди. Агент мусить досліджувати, які дії приноситимуть максимальну винагороду. В найцікавіших випадках дії можуть вплинути не лише на миттєву винагороду, а й на наступну ситуацію, тобто і на майбутні винагороди. Ці дві характеристики - спроба-помилка та винагорода з затримкою - є найважливішими особливостями навчання з підкріпленням. Основою ідеєю навчання з підкріпленням є здатність агента

розуміти найважливіші аспекти реальної проблеми, з якою він стикається крок за кроком в певному середовищі для досягнення мети. Агент мусить розуміти свій стан в середовищі. Також він має певну множину дій, якими він може впливати на середовище. Одним з викликів, що випливає в навчанні з підкріпленням є баланс між дослідженням та отриманням вигоди. Щоб отримати більшу винагороду, агент мусить надавати перевагу діям, які він вже спробував в минулому та визначив їх як ефективні.[12] Проте, щоб дослідити ці дії він мусить спробувати дії, які не використовував до цього. Він мусить застосувати ті дії, що вже знає, але, водночас, він мусить дослідити нові дії, щоб обирати кращі дії в майбутньому. Ця дилема показує, що не можна цілковито покладатись на жодну модель поведінки. Потрібно обрати спосіб динамічно підбирати модель поведінки агента. На рис. 5 зображено робочий цикл агента.

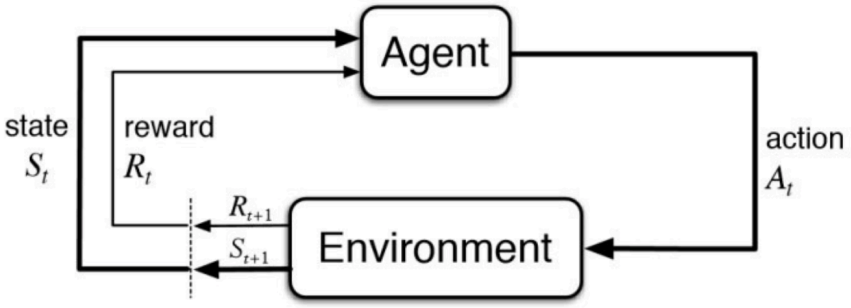

Рисунок 5 - Схема роботи агента

Висновки до розділу 1

Сучасні оркестратори є доволі розвиненою та технологічною відповіддю на бажання полегшити керування інфраструктурою програмних систем. Проте, досі вимагають ретельного контролю та розуміння, що є як джерелом незручностей так і джерелом помилок. Саме тому в даному розділі здійснено огляд навчання з підкріпленням, що має потенціал розв’язати задачу оркестрації в принципово новий спосіб.

Навчання з підкріпленням - це обчислювальний підхід до розуміння та автоматизації цілеспрямованого навчання та прийняття рішень. Його відрізняють від інших обчислювальних підходів шляхом акцентування уваги на засвоєнні агентом безпосередньої взаємодії з ним навколишнього середовища, не вимагаючи зразкового нагляду чи повних моделей навколишнього середовища.

Навчання з підкріпленням використовує формальну основу процесів прийняття рішень Маркова, що визначає взаємодію між навчальним агентом та його середовищем з точки зору станів, дії та винагороди. Цей фреймворк має бути простим способом представлення суттєвих особливостей проблем штучного інтелекту. Ці особливості включають відчуття причин та наслідків, відчуття невизначеності та існування явної цілі.[12]

РОЗДІЛ 2 ОПИС РОЗРОБЛЕНОЇ СИСТЕМИ

2.1 Опис середовища та параметрів системи

Середовищем агента є кластер, що складається з машин(воркерів), кожна з яких має певну кількість ресурсів на виконання запитів. Кожен запит характеризується складністю та часом виконання.

Приклад: один воркер має потужність 2.0 cpu. Це означає, що одночасно воркер може виконувати 10 запитів, кожен з яких потребує 0.2 cpu.

Якщо жоден воркер не може опрацювати запит через брак ресурсів - він очікує виконання. Час очікування на виконання є критичним показником для користувача.

Статичні параметри:

1) node_price - ціна одного воркера за секунду;
2) latency_price - ціна затримки;
3) success_request_price - ціна завершеного запиту;
4) failed_request_price - ціна незавершеного запиту;
5) capacity - потужність воркера;
6) latency_threshold - ліміт часу, що не вважається затримкою.

Параметри, на які агент має вплив:

1) node_count - кількість воркерів;
2) node_status - статус кожного воркера;
3) utilization - завантаження кожного воркера.

Можливі дії агента:

1) add_node - додати воркера;
2) remove_node_gracefully - видалити воркера, дочекавшись завершення запитів;
3) remove_node - видалити воркера негайно;
4) do_nothing - не змінювати конфігурацію кластера.

Показники, що полегшують моніторинг системи:

1) nodes_active - кількість активних воркерів;
2) success_request_count - кількість завершених запитів;
3) failed_request_count - кількість незавершених запитів;
4) total_latency - сумарна затримка;
5) total_processing_time - сумарний час запитів;
6) failed_request_percent - відсоток незавершених запитів;
7) node_utilization - завантаження воркера;
8) total_utilization - завантаження сумарне;
9) avg_utilization - середнє завантаження.

Функція винагороди включає:

1) винагороду за завершений запит;
2) штраф за незавершений запит;
3) штраф за працюючого воркера;
4) штраф за затримку.

2.2 Опис технологій, використаних для розробки

Для створення агента та його навчання використано:

1) мову програмування Python;
2) фреймворк RLGlue;
3) середовище Jupiter notebook.

Натренована з допомогою фреймфорку RLGlue модель(агент) експортуєтись в вигляді файлу. Далі, з допомогою засобів та бібліотек мов програмування, агент можна завантажувати в програму, що працює в фоновому режимі. Інша фонова програма генерує запити та відправляє їх

агенту. Агент автоматично розподіляє запити на вільні ноди. Якщо вільних нод не знайдено - запит стає в чергу. Кожні N секунд агент мусить обрати дію з набору можливих. В перспективі, для обміну інформацією між агентом та воркерами може бути використано технологію hazelcast. На рисунку 6 схематично зображено архітектуру агента. На рис. 6 зображено архітектуру системи.

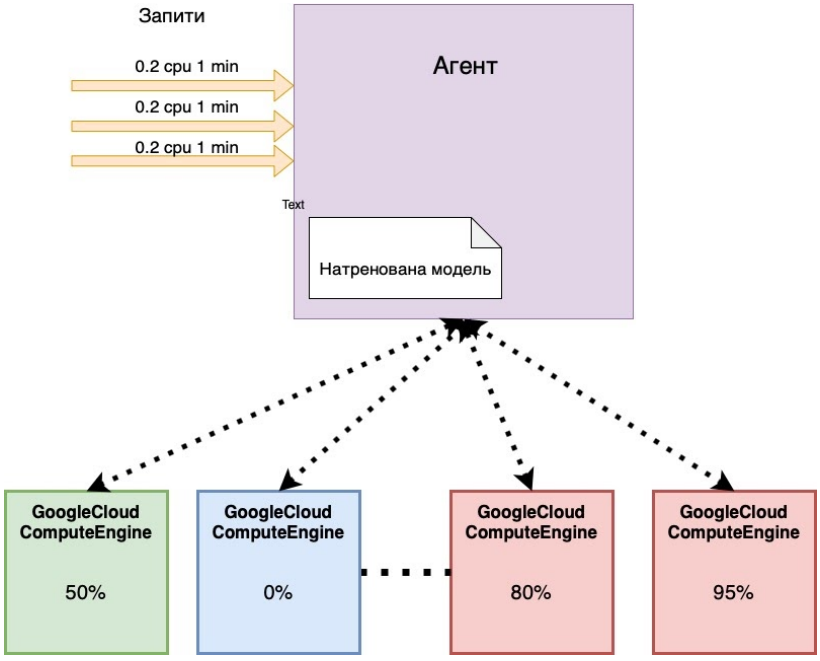

Рисунок 6 - Архітектура системи

Висновки до розділу 2

В даному розділі детально описано середовище агента, функцію винагороди та множину дій. Також наведено архітектуру системи в цілому та інструменти, використані для розробки.

РОЗДІЛ 3 ОПИС ПРОЦЕСУ ПІДГОТОВКИ АГЕНТА ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ

3.1 Основні компоненти агента

Основними складниками агента є:

1. Нейронна мережа для обрахунку action-value замість state-value.
2. Adam, використаний для тренування даної НМ.
3. Буфери відтворення досвіду.
4. Вибір дій з допомогою Softmax.
5. Expected Sarsa.

3.2 Action-value мережа

Системи навчання з учителем, засновані на глибоких нейронних мережах, тепер є основним рішенням для класифікації зображень, розпізнавання мови та обробки природної мови. Частина причин пов’язана з різким збільшенням даних, доступних для навчання. Але щоб насправді скористатися цим збільшенням обсягу даних та обчислень, потрібні були вдосконалення існуючих методів. Кілька простих стратегій оптимізації значно полегшили навчання мереж та допомогли пришвидшити обчислення. Як і багато методів машинного навчання, нейронні мережі навчаються за допомогою ітераційного процесу. Ця процедура повинна мати певну вихідну точку. Вибір цієї відправної точки може зіграти велику роль у роботі нейронної мережі. Як відомо, градієнтний спуск ітеративно перемістить вагу до найближчої нерухомої точки. Але функції втрати нейронної мережі не є настільки простими. Якщо ми починаємо з цієї майже рівної області, може бути важко досягти будь-якого прогресу з градієнтним спуском, оскільки градієнт близько нуля. Якщо замість цього ми почнемо всередині маленької

чаші, то ми зможемо швидко знайти локальний оптимум. Однією простою, але ефективною стратегією ініціалізації є випадковий відбір вихідних ваг із нормального розподілу з невеликою дисперсією. Таким чином, кожен нейрон має різний вихід від інших нейронів у своєму шарі. Це забезпечує більш різноманітний набір потенційних можливостей. Зберігаючи варіанти малими, ми гарантуємо, що вихід кожного нейрона знаходиться в тому ж діапазоні, що і його сусіди. Після того, як ми вибрали початкову точку для нашої мережі, ми починаємо поступово вносити невеликі вдосконалення ваг за допомогою кроків стохастичного градієнтного спуску.[12]

Іншим способом вдосконалення навчання є розгляд більш досконалих механізмів оновлення. Дві загальні стратегії полягають у використанні методу моменту, який також називають адаптацією розміру імпульсу та векторного кроку. Якщо останні оновлення мають суперечливі вказівки, це вбиває імпульс. Зміна імпульсу мало вплине на оновлення, і ми зробимо регулярний крок градієнтного спуску. Імпульс прискорює навчання, тобто він швидше доходить до нерухомого стану. Іншим потенційним поліпшенням є використання окремого розміру кроку для кожної ваги в мережі. Поки що ми говорили лише про глобальний скалярний розмір кроку. Загальновідомо, що це проблематично, оскільки це може спричинити оновлення, яке є занадто великим для деяких ваг та замалим для інших ваг. Адаптація розмірів кроків для кожної ваги на основі статистичних даних про навчальний процес на практиці призводить до набагато кращих результатів. Замість оновлення за допомогою скалярної константи, існує вектор розмірів кроків, проіндексований t, щоб вказати, що він може змінюватися на кожному кроці часу. Кожен вимір градієнта масштабується відповідно до його розміру кроку замість загального розміру кроку.

Ми використовує нейронну мережу для апроксимації функції action-value. Основна відмінність між апроксимацією функції значення state-value та функції action-value за допомогою нейронної мережі полягає в тому, що в першому вихідний рівень включає лише одну одиницю, тоді як у другому вихідний рівень включає стільки одиниць, скільки кількість дій.

Adam - це модифікація стохастичного градієнтного спуску, що містить в собі дві концепції: адаптивний розмір кроку навчання та момент.

На рис 7. зображено формули для обрахунку.

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$
$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2$$
$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-\beta_1^{t+1}}$$
$$\hat{v}_w = \frac{v_w^{(t+1)}}{1-\beta_2^{t+1}}$$
$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

Рисунок 7 - Обрахунок моментів та ваг для методу Adam.

Адам обрано через те, що він сходиться швидше за стохастичний градієнтний спуск та потребує менших налаштувань.

3.3 Буфери відтворення досвіду

Якби ми спробували це зробити в безперервному домені стану, у нас би було нескінченна кількість пар дія-стан. Або ще гірше, ми не побачимо одного і того ж стану двічі. Один із варіантів - навчити модель, використовуючи всі нові знання про наближення функцій. Наприклад, ми можемо навчити функцію параметризації, яка вводить стани та дії та виводить наступні стани та винагороди. На жаль, навіть незначні помилки в такій моделі динаміки навчання викликають серйозні проблеми. Залишається

відкритим питання ресурсу, як ефективно використовувати моделі навчання в агенті. Відтворення досвіду - це простий спосіб спробувати отримати переваги Dyna. Основна ідея полягає в тому, щоб зберегти буфер досвіду і нехай дані будуть моделлю.[13] Ми відбираємо досвід роботи з цим буфером та оновлюємо функцію значення за допомогою цих зразків подібно до того, як проводимо вибірку з моделі та оновлюємо значення в Dyna. Ключовим параметром відтворення досвіду є розмір буфера, який досвід зберігати та скільки оновлень виконувати за крок. Коли агент взаємодіє зі світом, він отримує стан, дію, винагороду, наступний стан. Ми додаємо цей досвід до буфера, який ми будемо називати буфером відтворення досвіду. Продовжуючи взаємодію зі світом, ми додаємо більше зразків до цього буфера, поки врешті не заповнимо його. Коли це трапляється, ми можемо вибрати старіший досвід, який потрібно видалити та замінити новим. Нам також потрібно враховувати розмір нашого буфера. Дозволяючи буферу бути справді великим, ми можемо пам’ятати потенційно корисні переходи з багаторазових кроків. Однак агент має практичні обмеження. Нам потрібно буде врахувати обсяг пам'яті, який використовується великими буферами, та будь-які обчислювальні наслідки зберігання та доступу до великих буферів. Використання одного зразка з буфера відтворення досвіду може спричинити шумне оновлення. Натомість ми можемо використовувати кілька зразків з буфера, щоб створити усереднене оновлення для зменшення цього шуму. Невелика колекція зразків із більшого буфера називається міні-батчем. Оновлення міні-батчем передбачає усереднення оновлень по K випадкових вибірках у міні-батчі.[12]

Повторний досвід дозволяє агенту краще використовувати свої дані і таким чином бути більш ефективним. Крім того, це гарантує, що агент оновлюється в багатьох частинах навколишнього середовища.

3.4 Expected Sarsa

Розглянемо алгоритм навчання, подібний до Q-навчання, за винятком того, що замість максимального для наступних пар action-value він використовує очікуване значення, беручи до уваги, наскільки кожна дія є відповідно до поточної політики. На рис. 8 наведено алгоритм із правилом оновлення.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big]$$
$$\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big],$$

Рисунок 8 - Правило оновлення Expected Sarsa

Враховуючи наступний стан, St + 1, цей алгоритм рухається детерміновано в тому ж напрямку, що і Сарса в очікуванні, і відповідно він називається Expected Sarsa. Схема резервного копіювання зображена праворуч на рисунку 9. Очікувана Сарса є більш складною обчислювально, ніж Сарса, але натомість вона усуває дисперсію через випадковий вибір Ат + 1. Враховуючи той же обсяг досвіду, ми могли б очікувати, що він буде працювати трохи краще, ніж Сарса, і насправді він це робить. На рисунку 9 наведено підсумкові результати для завдання ходіння по скелі з очікуваною сарсою порівняно з сарсою та Q-навчанням. Очікувана Сарса зберігає значну перевагу Сарси перед Q-навчанням щодо цієї проблеми. На рис. 9 зображено графіки сум функцій винагороди для Q-learning, Expected Sarsa та Sarsa.

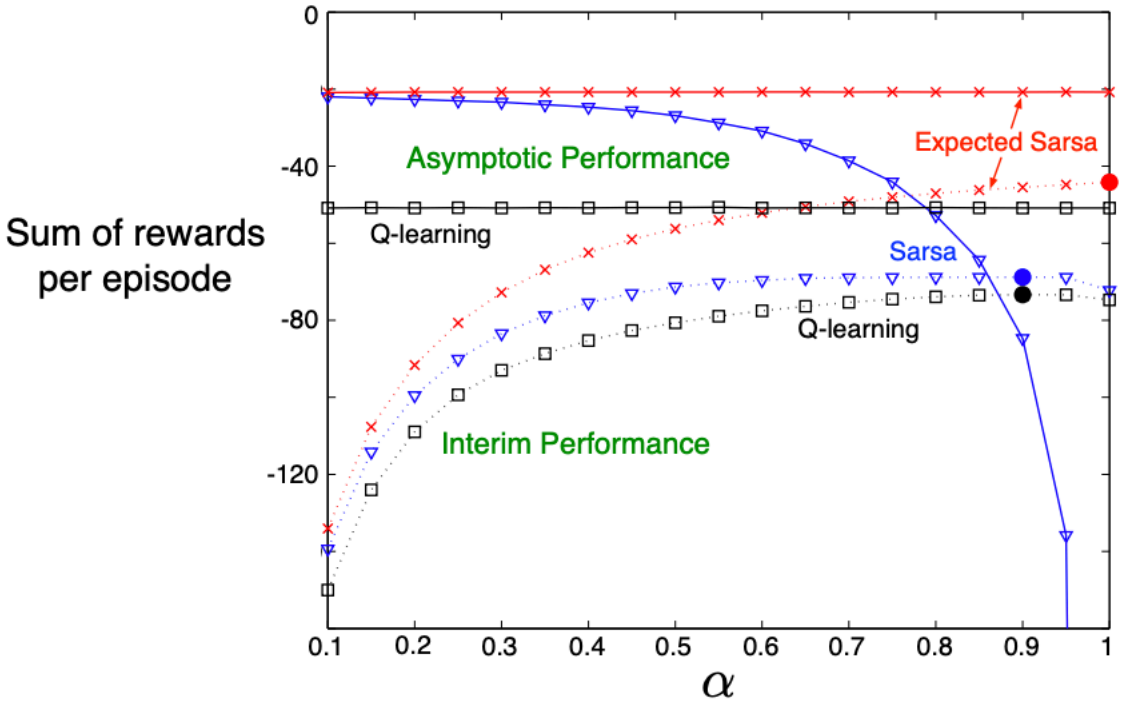

Рисунок 9 - Проміжне та асимптотичне виконання методів управління TD на задачі прогулянки по скелі

Крім того, очікувана Sarsa демонструє значне покращення порівняно з Sarsa у широкому діапазоні значень для параметра кроку-розмір альфа. У ході скелею переходи станів є детермінованими, і вся випадковість походить від політики. У таких випадках очікувана Сарса може безпечно встановити альфа = 1, не зазнаючи жодних погіршень асимптотичної продуктивності, тоді як Сарса може добре працювати в довгостроковій перспективі при невеликому значенні альфа, при якому короткочасна ефективність погана.

У цьому та інших прикладах є послідовна емпірична перевага Очікуваної Сарси над Сарсою. У цих результатах ходьби по скелі Очікувана Sarsa була використана на політиці, але загалом вона може використовувати політику, відмінну від цільової політики PI, для формування поведінки, і в цьому випадку вона стає алгоритмом політики PI. Наприклад, припустимо, що PI є жадібною політикою, тоді як поведінка є більш дослідницькою; тоді Очікувана Сарса - це саме Q-навчання. У цьому сенсі Очікувана Сарса включає та узагальнює Q-навчання, одночасно надійно покращуючи Сарсу.

За винятком невеликих додаткових обчислювальних витрат, очікувана Сарса може повністю домінувати в обох інших більш відомих алгоритмах управління TD.[12]

3.5 Softmax

Суттєвою перевагою використання softmax для вибору політики є те, що вона враховує значення дій. Це означає, що дія з помірним значенням має більший шанс бути обраною ніж дія з низьким значенням. Це сильно відрізняється від ”жадібної” політики, що не враховує значення дії дослідження при виборі, а обирає випадковим чином.

3.6 Вибір функції винагороди

Функції винагороди описують, як агент "повинен" поводитися. Іншими словами, вони мають зміст, який визначає, що ви хочете, щоб виконав агент.

Пошук функції винагороди є найскладнішою частиною проблеми, вона тісно пов’язана з тим, як ми вказуємо простір стану.

На абстрактному рівні навчання з підкріпленням повинне уникнути "правильних і неправильних" результатів. Але ми бачимо зараз, що навчання з підкріпленням просто перекладає відповідальність з вчителя на функцію винагороди. Один із методів називається зворотним навчанням з підкріпленням або "навчання учнівству", яке генерує функцію винагороди, яка відтворюватиме спостережену поведінку.

На рис. 10 зображено обрану функцію винагороди агента.

$$
\begin{aligned}
reward = \\
&- node\_active * node\_price \\
&+ success\_request\_count * request\_price \\
&- failed\_request\_count * failed\_request\_price \\
&- total\_latency * latency\_price
\end{aligned}
$$

Рисунок 10 - Функція винагороди агента.

Для підбору ефективної комбінації коефіцієнтів було проведено кілька експериментів. На рис. 11 зображено графік залежності значення функції винагороди з параметрами node_price = 50, request_price = 5, failed_request_price = 10, latency_price = 3.

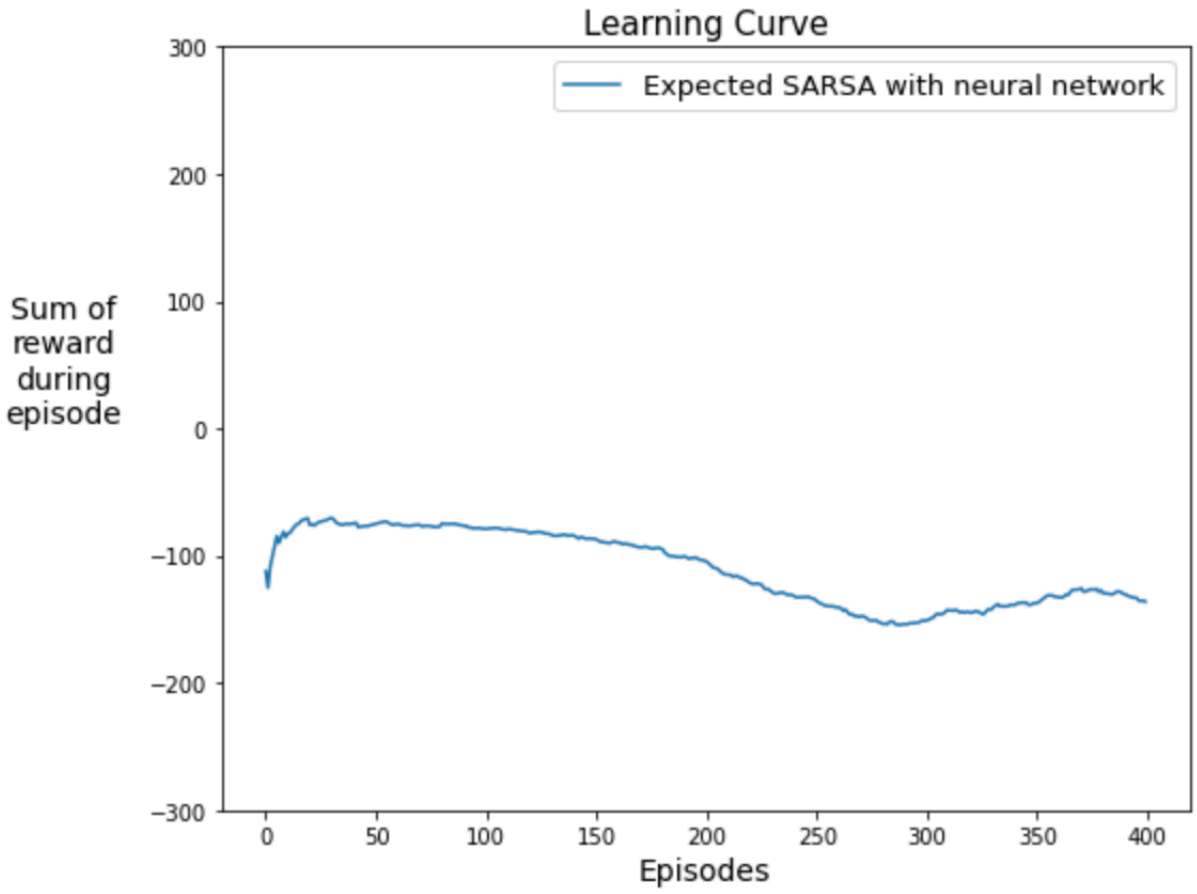

Рисунок 11 - Графік залежності значення функції винагороди з параметрами node_price = 50, request_price = 5, failed_request_price = 10, latency_price = 3.

З графіку видно, що зі збільшенням кількості епох результат лише погіршеється, тому спробуємо інші коефіцієнти. На рис. 12 зображено графік залежності значення функції винагороди з параметрами node_price = 100, request_price = 5, failed_request_price = 5, latency_price = 5.

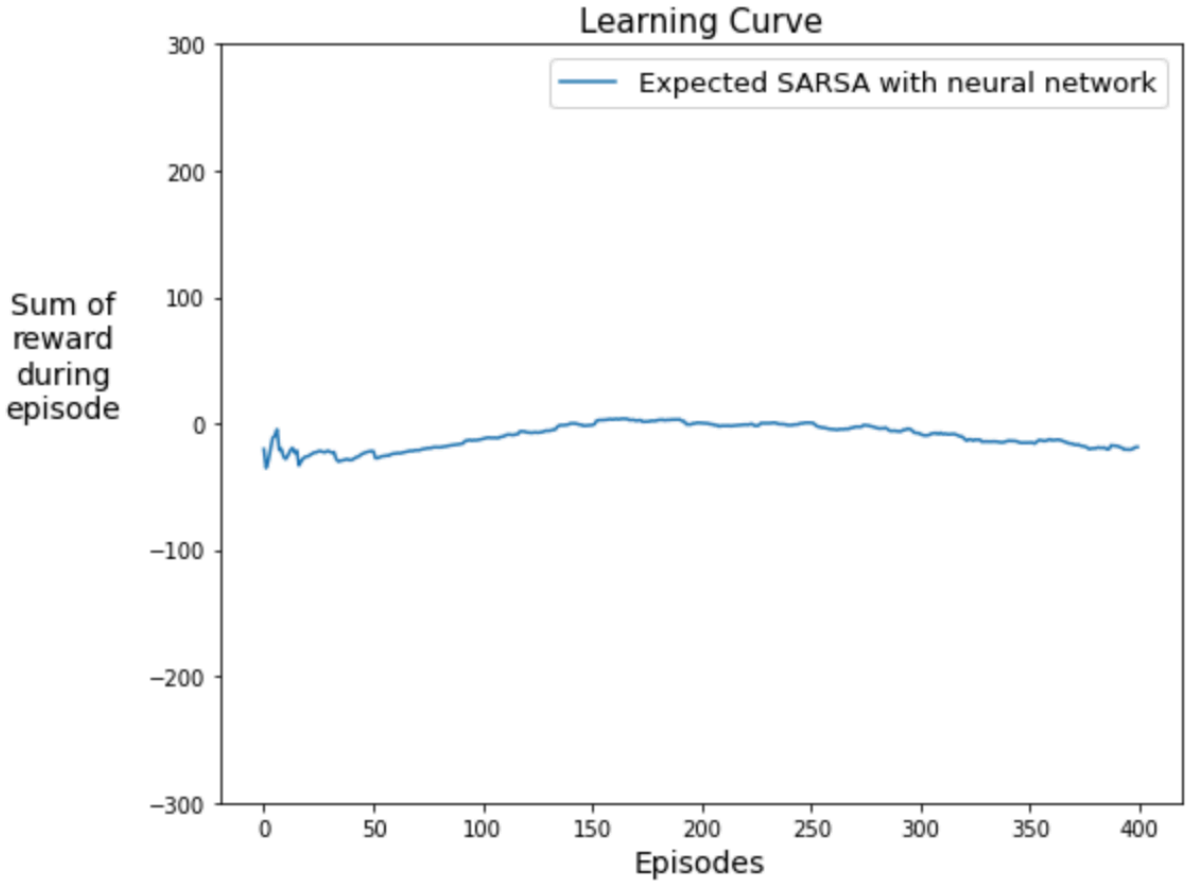

Рисунок 12 - Графік залежності значення функції винагороди з параметрами node_price = 100, request_price = 5, failed_request_price = 5, latency_price = 5.

Дана комбінація працює краще за попередню, тому спробуймо підняти всі коефіцієнти, крім node_price. На рис. 13 зображено графік залежності значення функції винагороди з параметрами node_price = 100, request_price = 10, failed_request_price = 10, latency_price = 10.

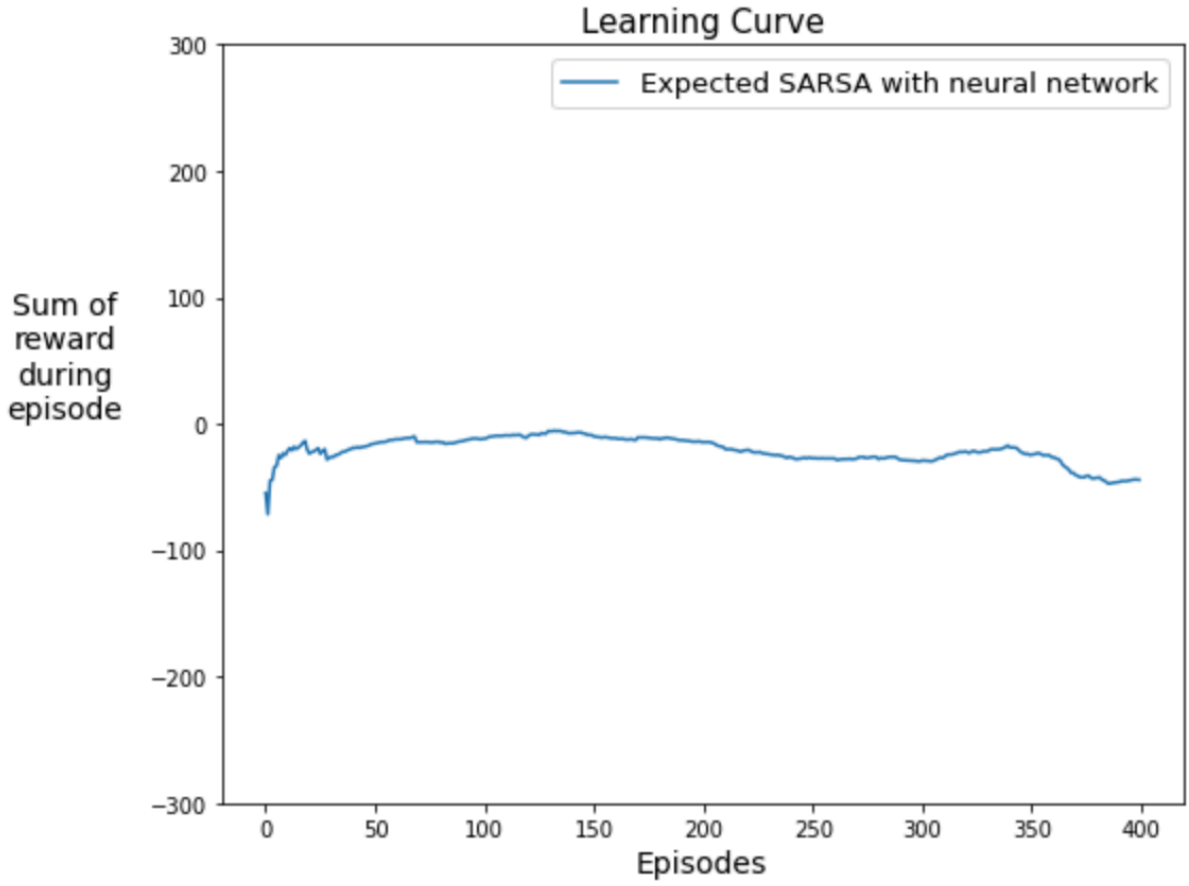

Рисунок 13 - Графік залежності значення функції винагороди з параметрами node_price = 100, request_price = 10, failed_request_price = 10, latency_price = 10.

Спробуймо збільшити винагороду для успішно виконаних запитів до 100. На рис. 14 зображено графік залежності значення функції винагороди з параметрами node_price = 100, request_price = 100, failed_request_price = 10, latency_price = 10.

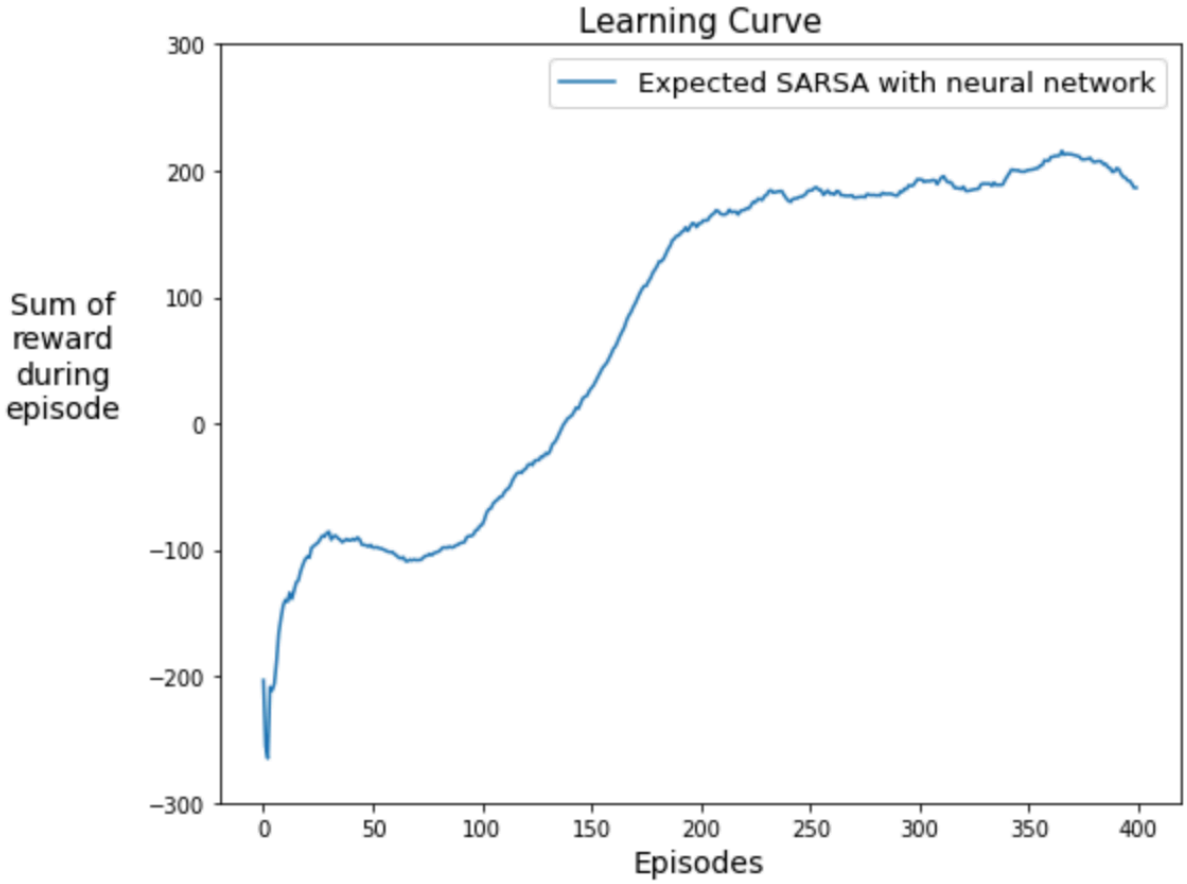

Рисунок 14 - Графік залежності значення функції винагороди з параметрами node_price = 100, request_price = 100, failed_request_price = 10, latency_price = 10.

З графіку видно, що функція вже показує досить хороші результати, варто спробувати ще підняти вагомість показника failed_request_price до 50. На рис. 15 зображено графік залежності значення функції винагороди з параметрами node_price = 100, request_price = 100, failed_request_price = 50, latency_price = 10.

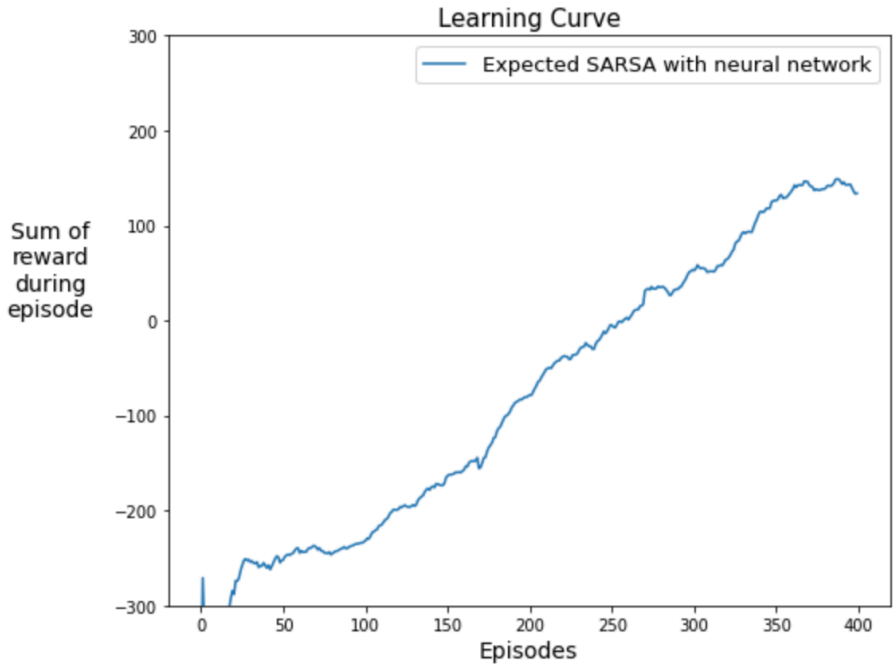

Рисунок 15 - Графік залежності значення функції винагороди з параметрами node_price = 100, request_price = 100, failed_request_price = 100, latency_price = 10.

3.7 Аналіз результатів роботи агента

Конфігурація агента:

- num_hidden_units - 256;
- state_dim - 8;
- num_actions - 4;
- step_size - 1e-3;
- beta_m - 0.99;
- beta_v - 0.998;
- epsilon - 1e-8;
- replay_buffer_size - 55000;
- minibatch_sz - 8;
- num_replay_updates_per_step - 4;
- gamma - 0.98;

- tau - 0.002;
- num_episodes - 400.

Коефіцієнти функції винагороди:

- node_price = 100
- request_price = 100
- failed_request_price = 100
- latency_price = 10

На рис. 16 наведено графік росту значення функції винагороди з кількістю епох

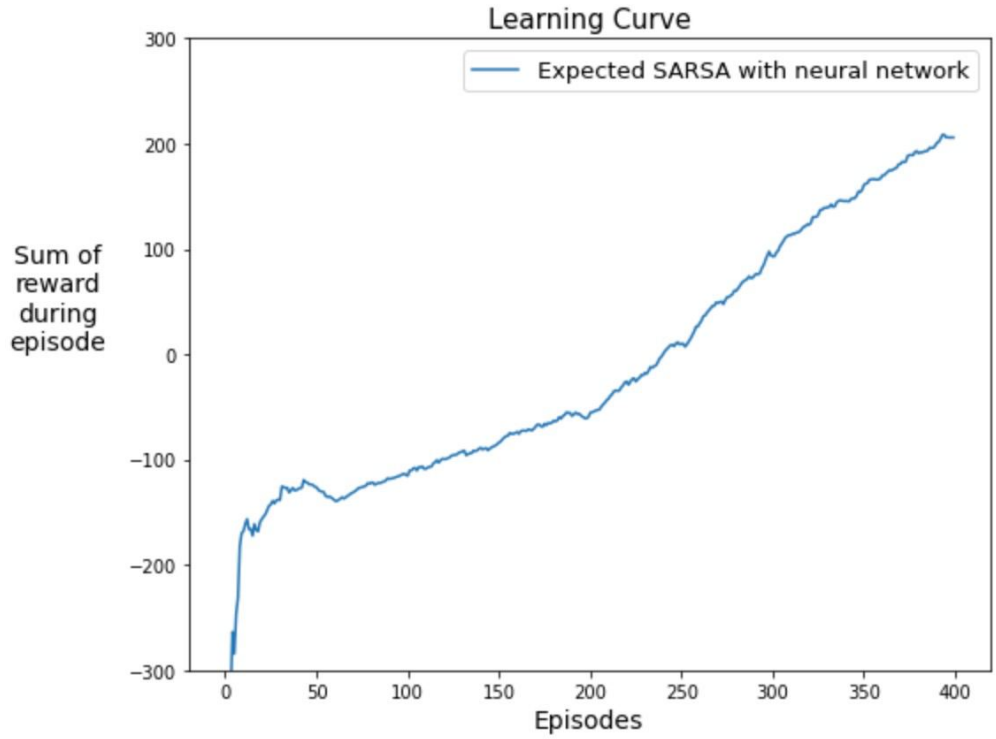

Рисунок 16 - Графік зміни значення функції винагороди(вісь y) з ростом кількості епізодів(вісь x), node_price = 100, request_price = 100, failed_request_price = 100, latency_price = 10.

З графіку еволюції агента яскраво видно, що спочатку агент працює невдало, проте, після 250 епох, дана конфігурація агента показує досить гарні результати. На рис. 17 можна побачити тест роботи агента та його реакцію.

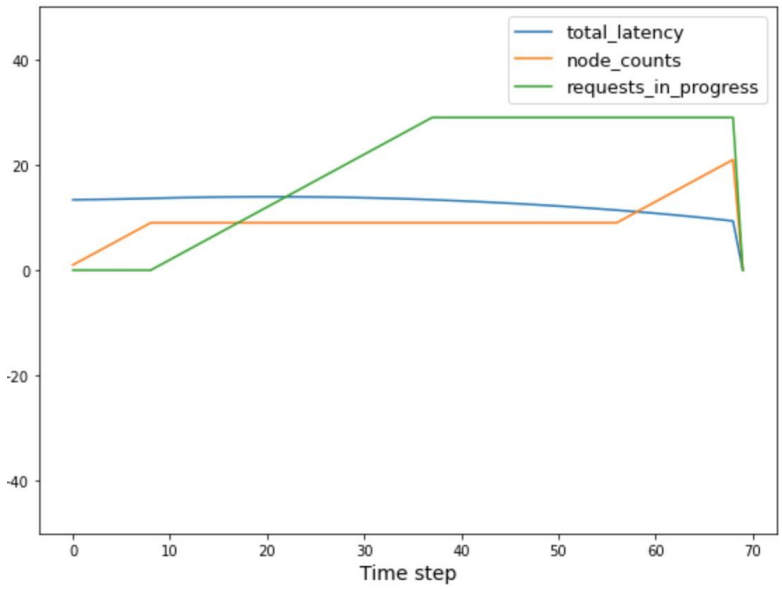

Рисунок 17 - Графік реакцій агента на збільшення кількості запитів(200 епох, буфер 55000)

Агент реагує на збільшення кількості запитів, збільшуючи кількість екземплярів сервісу. В наступному експерименті кількість епох збільшено до 400. На рис. 18 зображено результати експерименту.

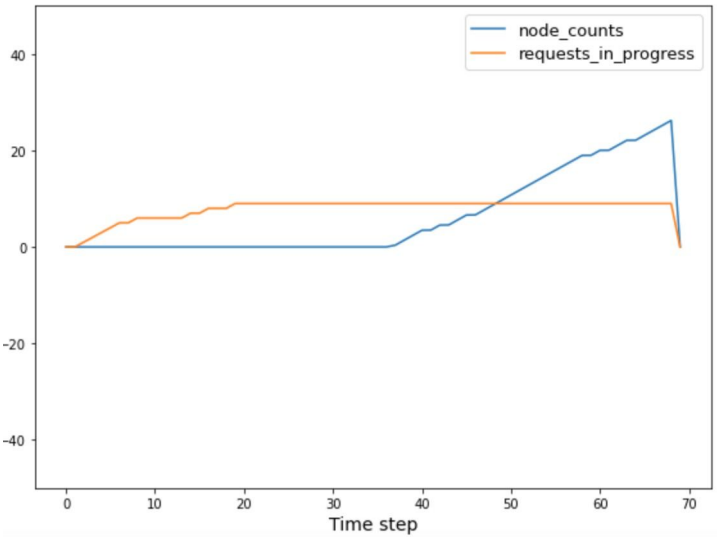

Рисунок 18 - Графік реакцій агента на збільшення кількості запитів(400 епох, буфер 55000)

Проте, при недостатньому розмірі буфера, агент працює занадто повільно. На рис. 19 можна побачити некоректну поведінку агента.

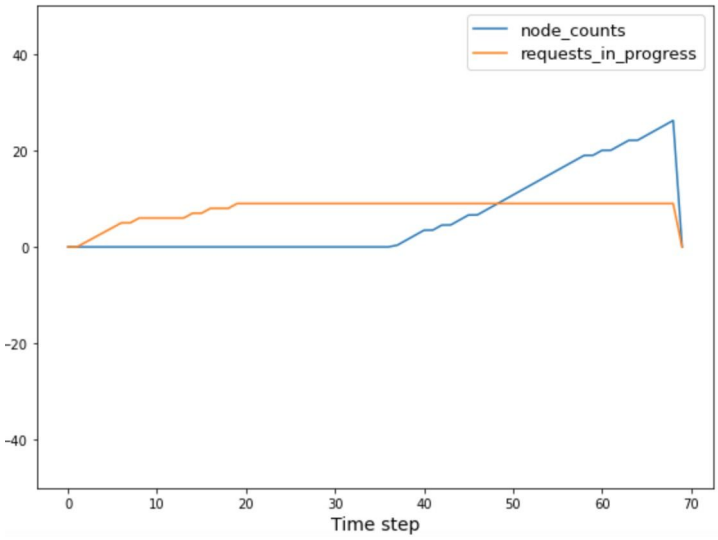

Рисунок 19 - Графік реакцій агента на збільшення кількості запитів(400 епох, буфер 20000)

На рис. 20 зображено кінцевий графік розвитку агента.

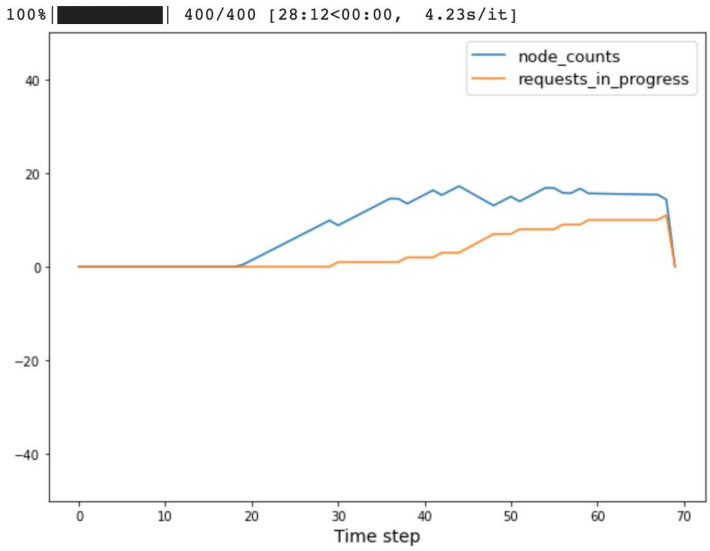

Рисунок 20 - Графік реакцій агента на збільшення кількості запитів(400 епох, буфер 20000)

Висновки до розділу 3

Цей розділ присвячено технічним особливостям програмного рішення та аналізу результатів роботи. Архітектура агента є доволі складною, оскільки потребує попередніх навичок роботи з техніками машинного навчанням з учителем. Також достатньо багато уваги було приділено підбору ефективної функції винагороди. З результатів роботи агента видно, що за достатньої кількості епох та глибокого буфера досвіду агент добре реагує на динамічне середовище. Якщо буфер недостатнього об’єму або замало епох, то агент доволі повільно реагує на зміну навантаження.

РОЗДІЛ 4 РОЗРОБКА СТАРТАП ПРОЕКТУ

В даному розділі наведено маркетинговий аналіз стартап проекту. В межах цього етапу:

1) розробляється опис самої ідеї проекту та визначаються загальні напрями використання потенційного товару чи послуги, а також їх відмінність від конкурентів;
2) аналізуються ринкові можливості щодо його реалізації;
3) на базі аналізу ринкового середовища розробляється стратегія ринкового впровадження потенційного товару в межах проекту.

4.1 Опис ідеї проекту

Більш детальну інформацію про проект наведено в інформаційній карті проекту, у табл. 4.1.

Таблиця 4.1— Інформаційна карта проекту.

Назва проекту	orchtopus - система оркестрації в розподілених обчислювальних мережах на основі навчання з підкріпленням
Автор проекту	Міщук Андрій Романович
Коротка анотація	Основне призначення аналітичної системи – створення агента, здатного підлаштовуватись під динамічне навантаження на систему без втручання адміністратора.
Термін реалізації	12 місяців

Продовження табл. 4.1.

Необхідні ресурси	2 технічних спеціалістів (1 розробник моделей, 1 cloud DevOps), 2 ПК, профіль в Google Cloud, патент на ідею
Опис проблеми, яку вирішує проект	Необхідність ручного переналаштування оркестратора під зміну продукту.
Головні цілі та завдання проекту	Створити інструмент, що зможе реагувати на зміни автоматично.
Очікувані результати	Агент вміє швидко адаптуватись під нові умови.

У табл. 4.2 наводиться опис необхідної команди для розробки стартап проекту та основних функціональних обов'язків, необхідні вимоги досвіду та знань членів команди.

Таблиця 4.2 — Команда стартап-проекту

Керівник проекту, розробник моделей	Обов’язки: керування діяльністю проекту, створення, підтримка та покращення агента
DevOps	Обов’язки: Налаштування та підтримка кластера на базі Google Cloud Compute Engine, налаштування та підтримка CI/CD

Наступний пункт маркетингова стратегія та маркетинговий план стартапу, в якому послідовно проаналізовано та подано у вигляді таблиць:

1) зміст ідеї – табл. 4.1, яка дає цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів;

2) можливі напрямки застосування – табл. 4.2, а саме аналіз потенційних техніко-економічних переваг ідеї (чим відрізняється від існуючих аналогів та замінників) порівняно із пропозиціями конкурентів передбачає:

3) основні вигоди, що може отримати користувач товару (за кожним напрямком застосування) – табл. 4.3;

4) чим відрізняється від існуючих аналогів та замінників – табл. 4.4.

Таблиця 4.3 — Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Створення гнучкого оркестратора розподілених систем	Ринок IT інфраструктури	Зменшення витрат на підтримку систем. Ефективніша витрата коштів.

Таблиця 4.4 — Визначення сильних, слабких та нейтральних характеристик ідеї проекту.

Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			Слабка сторона	Нейтральна сторона	Сильна сторона
	Orchtopus	Docker Swarm	Kubernetes			
Ціна	0.10$/ cluster/ mo	$15/node/ mo	$0.10/ cluster/ mo			+
Рівень швидкодії	високий	високий	середній			+

Продовження табл. 4.4.

Рівень практичності результату	високий	високий	середній		+	
Рівень зручності інтерфейсу	високий	високий	високий			+
Додаткова інформація	є	є	є			+
Гнучкість системи	висока	середня	висока	+		

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

4.2 Технологічна здійсненність ідеї проекту

Далі визначені ринкові можливості, які будуть використовуватись під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту. У табл. 4.5 наведено технологічну

здійсненність проекту.

Таблиця 4.5 — Технологічна здійсненність ідеї проекту

Ідея проекту	Технологіїїї реалізації	Наявність технологій	Доступність технологій
Гнучкий оркестратор сервісів	RLGlue, Python, GCE	так	так

У табл. 4.6 проводиться аналіз попиту: наявність, обсяг, динаміка розвитку ринку.

Таблиця 4.6 — Попередня характеристика потенційного ринку стартап-проекту

Показники стану ринку (найменування)	**Характеристика**
Кількість головних гравців, од	2
Загальний обсяг продаж, грн/ум.од	200 000 грн
Динаміка ринку (якісна оцінка)	Стагнує
Наявність обмежень для входу (вказати характер обмежень)	-
Специфічні вимоги до стандартизації та сертифікації	-
Середня норма рентабельності в галузі (або по ринку), %	15

Надалі у табл. 4.7 визначено потенційні групи клієнтів, їх

характеристики, та сформовано орієнтовний перелік вимог до товару для кожної групи.

Таблиця 4.7 — Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	**Цільова аудиторія (цільові сегменти ринку)**	**Відмінності у поведінці різних потенційних цільових груп клієнтів**	**Вимоги споживачів до товару**
Зменшення витрат на підтримку інфраструктури	Архітектори програмних продуктів	Гнучка поведінка оркестратора в динамічному середовищі	Google cloud в якості провайдера послуг

4.3 Фактори загроз

Після визначення потенційних груп клієнтів проведено аналіз ринкового середовища: складено таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають, зокрема у табл. 4.8 наведено фактори загроз, табл. 4.9 містить фактори можливостей. Фактори в таблицях наведені в порядку зменшення значущості.

Таблиця 4.8 — Фактори загроз

Фактор	**Зміст загрози**	**Можлива реакція компанії**
Конкуренція	Вихід на ринок продуктів з кращими характеристиками	Вдосконалення технічних моментів власного продукту, акції, зниження цін для утримання цільової аудиторії
Зовнішній вплив економічної ситуації	Економічна нестабільність	Знизити вартість придбання або додати можливості скористатись обмеженою безкоштовною версією
Невідповідність умовам провайдера послуг	Небезпечно підв’язувати продукт під єдиного провайдера - Google Cloud	Інтеграція з іншими гравцям ринку: AWS, Azure

Таблиця 4.9 — Фактори можливостей

Фактор	Зміст можливості	Можлива реакція компанії
Співпраця з відомими компаніями	Можливість протестувати систему на більших кластерах	Покращення системи
Залучення open-source розробників	Здобуття впізнаваності на ринку	Збільшення цільової аудиторії

Далі проведено аналіз пропозиції: визначено загальні риси конкуренції на ринку, що наведені у табл. 4.10.

Таблиця 4.10 — Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	**В чому проявляється дана характеристика**	**Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)**
Тип конкуренції	Олігополія	Модернізація функціоналу.
За рівнем конкурентної боротьби	Національний	Доступ до функцій мають лише користувачі певного регіону
За галузевою ознакою	Внутрішньогалузева	Пропозиція більш вигідних умов
Конкуренція за видами товарів	Товарно-видова	Розробка додаткового функціоналу для задоволення конкретного бажання клієнта
За характером конкурентних переваг	Нецінова	Удосконалення моделі, розширення зони керування агента
За інтенсивністю	Немарочна	Реклама та знижки

Після аналізу конкуренції проведено більш детальний аналіз умов конкуренції в галузі, який наводиться у табл. 4.11.

Таблиця 4.11 — Аналіз конкуренції в галузі за М. Портером

Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Docker Swarm, Kubernetes	Інші програмні продукти для підтримки інфраструктури	Cloud провайдери	Юридичні особи	Безкоштовні версії
Висока інтенсивність конкуренції	Можливості входу в ринок є. Необхідне тестування на великих масштабах. Строки виходу – менше 1 року.	Ціна за кластер визначається умовами співпраці.	Клієнт орієнтований на практичність результату та оптимальну цінову політику.	-

На основі аналізу конкуренції, проведеного у табл. 4.11, а також із урахуванням характеристик ідеї проекту, наведених у табл. 4.4, вимог споживачів до товару – табл. 4.7 та факторів маркетингового середовища, наведених відповідно у табл. 4.8 та табл. 4.9, визначено та обґрунтовано перелік факторів конкурентоспроможності. Аналіз оформлений у табл. 4.12.

Таблиця 4.12 — Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	**Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)**
Ціна	Система має безкоштовну версію, що дозволяє ознайомитись з продуктом.
Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
Легке використання	Не потрібно витрачати багато часу для отримання результату.
Інноваційний характер виробництва	Відсутність схожого функціоналу на сьогоднішній день.
Орієнтація на маркетингову концепцію	Продукт орієнтований на взаємодію з клієнтом

За визначеними факторами конкурентоспроможності, що наведені у табл. 4.12, проведено аналіз сильних та слабких сторін стартап-проекту. Детальний опис відображено у табл. 4.13.

Таблиця 4.13 — Порівняльний аналіз сильних та слабких сторін стартап-проекту

Фактор конкурентоспроможності	**Бали** 1-20	**Рейтинг товарів-конкурентів**						
		-3	-2	-1	0	1	2	3
Ціна	18	*						
Легке використання	20		*					
Інноваційний характер виробництва	8				*			
Орієнтація на маркетингову концепцію	7			*				

4.4 SWOT-аналіз стартап-проекту

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу, який наводиться у табл. 4.14.

Таблиця 4.14 — SWOT-аналіз стартап-проекту

Сильні сторони:	**Слабкі сторони:**
Цінова політика Унікальний практично орієнтований функціонал Швидкість роботи Легкість використання	Мала база клієнтів Недостатність тестування на великих системах
Можливості:	**Загрози:**
Бурхливий розвиток систем, що базуються на навчанні з підкріпленням	Економічна нестабільність Швидкий залежність від постачальника послуг

На основі SWOT-аналізу розроблено альтернативи ринкової поведінки, що приводяться у табл. 4.15.

Таблиця 4.15 — Альтернативи ринкового впровадження стартап-проекту

Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
Підписання договору з постачальниками послуг, побудова якісної моделі	Необхідна велика кількість ресурсів для отримання якісної моделі. Отримання цих ресурсів досить проблематичне, тому що необхідно залучати інвесторів.	Із ростом складності моделі росте строк її реалізації. Тому можливе відхилення від запланованого строку (12 місяців) на 6-8 місяців вперед.

Продобження табл. 4.15.

Співробітництво з відомими компаніями.	Необхідні математики та інші співробітники з досить високою кваліфікацією, тому що робота буде вестися з складними технологіями, на основі яких необхідно побудувати достатньо якісну модель	Строки реалізації залежать від досвідченості та кваліфікації співробітників. При достатньому рівні – 12 місяців. В разі високого рівня проблемності даних, строки можуть бути збільшені на 2-4 місяці.

Обрана альтернатива – 2: знайти кваліфікованих співробітників у наших умовах простіше, ніж інвесторів. Строки реалізації для даної альтернативи найменші.

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів, що наведені у табл. 4.16.

Таблиця 4.16 — Вибір цільових груп потенційних споживачів

Опис профілю цільової групи потенційних клієнтів	**Готовність споживачів сприйняти продукт**	**Орієнтовний попит в межах цільової групи (сегменту)**	**Інтенсивність конкуренції в сегменті**	**Простота входу у сегмент**
Юридичні особи, переважно ІТ установи	висока	високий	сильна	просто

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку, яка наведена у табл. 4.17.

Таблиця 4.17 — Визначення ключових переваг концепції потенційного товару

Обрана альтернатива розвитку проекту	**Стратегія охоплення ринку**	**Ключові конкурентоспроможні позиції відповідно до обраної альтернативи**	**Базова стратегія розвитку***
Альтернатива №2	Визначити потреби групи, розробити відповідно до них стратегії приваблення клієнтів та маркетингової комукації	Цінова політика, універсальність продукту (миттєве практичне застосування)	Стратегія диференціації

Наступним кроком є вибір стратегії конкурентної поведінки, що наведена у табл. 4.18.

Таблиця 4.18 — Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	**Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?**	**Чи буде компанія копіювати основні характеристики товару конкурента, і які?**	**Стратегія конкурентної поведінки***
«Першопроходець»	Шукати нових та відвойовувати існуючих	Ні	Стратегія заняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника (стартап- компанії) та до продукту, згідно табл. 4.7, а також в залежності від обраної базової стратегії розвитку, що наведена у табл. 4.17 та стратегії конкурентної поведінки – табл. 4.18, була розроблена стратегія позиціонування, яка наведена у табл. 4.19. Вона полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.19 — Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	**Базова стратегія розвитку**	**Ключові конкурентоспроможні позиції власного стартап-проекту**	**Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)**
Легкість розуміння Точність агента	Стратегія диференціанції	Позиція на основі порівняння фірми з товарами конкурентів;	Економія часу; Зручність застосування; Практичність результату

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 4.20 наведені результати попереднього аналізу конкурентоспроможності товару.

Товар займатиме ринковий сегмент, який відповідає цільовим категорія, що наведені у табл. 4.16.

Найкращі інструменти маркетингу – маркетингова комунікація, реклама.

Таблиця 4.20 — Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
Швидкість отримання результату	Швидка адаптація до змін програм	Відсутність необхідності людського втручання
Зручність користування	Не потрібно щоразу налаштовувати та вимірювати продуктивність системи	Агент сам реагує на зміну середовища

Нижче у табл. 4.21 наведена розроблена трирівнева маркетингова модель товару: уточнені ідея продукту та послуги, його фізичні складові, особливості процесу його надання.

Таблиця 4.21 — Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Швидка адаптація до змін програм		
II. Товар у реальному виконанні	Властивості/ характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
III. Товар із підкріпленням	Автоматизована реакція на зміни системи		
	Якість: високий показник використання ресурсів та низька затримка		
	Пакування відсутнє		
	До продажу		
	Після продажу		

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення. Опис системи збуту наведено у табл. 4.22.

Таблиця 4.22 — Визначення ключових переваг концепції потенційного товару

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Цільові клієнти – юридичні особи	Встановлення контактів із споживачами і підтримання їх Формування попиту і стимулювання збуту Розробка і реалізація програм з підтримки лояльності клієнтів	Один (від виробника одразу споживачу)	Прямий канал збуту до споживача, збільшити базу постачальників, мінімізувати збутові витрати (після запуску проету – зменшити ставку робітникам), розвиток маркетингового спілкування із споживачем

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів, яка наведена у табл. 4.23.

Таблиця 4.23 — Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення
Цільові клієнти – юридичні особи	Internet	Позиція на основі порівняння фірми з товарами конкурентів; Відмінні особливості споживача	Створення репутації фірмі — виробнику чи посереднику; збільшення чистого прибутку та рентабельності фірми

Варто відмітити, що на даний проект матиме місце достатній попит у зв’язку зі сприятливими економічними та соціальними умовами. На даний момент компанії, що займаються розробкою та наданням послуг зі схожим функціоналом, не готові до виходу на ринок конкурентоспроможної компанії, що дозволить певний час розвиватись та накопичувати базу клієнтів без перешкод. Оскільки даний проект буде поширюватись на міжнародному рівні, функціонал за невелику ціну, що в жодній мірі не вимагає фізичних та інших зусиль, матиме великий попит.

В якості альтернативи виходу на ринок обрано стратегію розробки моделі високоякісними професіоналами на невеликих кластерах, що дасть змогу виставити невисоку ціну за послугу. В результаті отримаємо велику базу клієнтів і зможемо підключати партнерів для залучення більш валідних та якісних даних. В такому випадку строки реалізації залежать тільки від кваліфікації співробітників та бажання якнайшвидше випустити продукт на ринок.

4.5 Бізнес модель

В таблиці 4.24 наведено шаблон бізнес-моделі, що дозволить якнайшвидше випустити продукт на ринок.

Таблиця 4.24 — Шаблон бізнес-моделі

<table>
<tr><th>Ключові партнери</th><th>Ключові види діяльності</th><th>Ціннісна пропозиція</th><th>Взаємостосунки з клієнтом</th><th>Споживацькі сегменти</th></tr>
<tr><td>Фінансово-економічні установи</td><td>Побудова прогнозів фінансово-екномічних процесів</td><td rowspan="3">Промо-акції, знижки</td><td>Клієнтоорієнтація</td><td rowspan="3">Юридичні особи</td></tr>
<tr><td rowspan="2"></td><td>Ключові ресурси</td><td>Канали збуту</td></tr>
<tr><td>Фінансово-укономічні установи</td><td>Інтернет</td></tr>
<tr><td colspan="3">Структура витрат:
Навчальні дані 30%, підтримка СППР 10%, заробітна плата 50%, просування ІАС 20%</td><td colspan="2">Поток доходів:
прибуток від продажу;
продаж моделі прийняття рішення</td></tr>
</table>

У табл. 4.25 наведено основні елементи фінансової моделі стартап-проект.

Таблиця 4.25 – Сукупні інвестиційні витрати на реалізацію стартап-проекту

Стаття витрат	**Сукупні витрати, тис. грн**
Загальні початкові витрати	
Проведення пошукових та прикладних досліджень	0
Розробка проектних матеріалів і ТЕО	10
Робоче проектування і прив'язка проекту	0
Витрати на придбання обладнання та устаткування та пристроїв	100
Витрати на придбання нематеріальних активів	20
Витрати на утримання обладнання та приміщень	-
Витрати на передвиробничі маркетингові дослідження	-
Витрати на створення збутової мережі	-
Витрати на просування та рекламу	50
Оплата юридичних послуг	10
Витрати на матеріальні ресурси	
матеріали	0
комплектуючі	0
сировина	0
Витрати на оплату праці команди стартапу	20 * 7 * 12 + 70
Разом	500

Висновки до розділу 4

У розділі 4 проведено аналіз можливості побудови старап-проекту на основі магістерської дисертації. Загалом, враховуючи усі можливості і фактори побудова стартапу можлива і потребує одного року для його реалізації.

ВИСНОВОК

Під час виконання досліджень магістерської дисертації обґрунтовано необхідність та доцільність покращення систем оркестрації в розподілених обчислювальних мережах. Проведено огляд сучасних рішень, їх архітектура, переваги та недоліки. Також наведено як основні принципи побудови систем на основі навчання з підкріпленням так і більш вузькоспрямовані техніки.

В ході дослідження було розроблено середовище, в якому генеруються запити та агент, що вміє додавати або забирати надлишкові ресурси(серевери). Така емуляція є досить близькою до реальних умов та допомагає краще зрозуміти поставлену задачу. Системи на основі навчання з підкріпленням показали свої переваги в задачах керування процесами і тому такий підхід до проблеми контролю ресурсів додає гнучкості системі оркестрації, що значно знижує рівень необхідного втручання адміністратора системи.

Агент складається з нейронної мережі, алгоритму Expected Sarsa та буферу відтворення досвіду. Нейронна мережа навчається з допомогою алгоритму Adam та використовується для обчислення кращої action-value комбінації. Буфер обміну допомагає ефективно емулювати “досвід” агента. Розроблений оркестратор показує хороші результати за наявності достатньо глибокого буферу досвіду(не менш ніж 50000) та достатньої кількості ітерацій(близько 400).

Також було здійснено стартап-аналіз проекту, розроблено шаблон бізнес-плану та виділено загрози, переваги та стратегії розвитку продукту. Даний проект є доволі інноваційним та перспективним, незважаючи на рівень розвитку класичних оркестраторів.

Дослідження полягало у тому щоб навчити моделі, методами машинного навчання, передбачати здатність позичальників до повернення взятих на себе коштів. Детальний опис методів та засобів для моделювання, обробки вхідних даних та способів оцінювання якості роботи моделей було

проведено у другому розділі. До методів які описані у цьому дослідженні належать: логістична регресія, метод опорних векторів та метод випадковий ліс. Також для якісної їх роботи у другому розділі було детально описано методи балансування вибірки, оскільки при його відсутності більшість класифікаторів не досягне поставленого результату.

З технічної точки зору, проект реалізовано мовою Python 3.7 з допомогою фреймворку RLGlue.

Дане дослідження відповідає всім поставленим завданням, та має перспективи розвитку, наприклад інтеграція агента в обгортку на базі Go, розгортання в якості рушія для Kubernetes, навчання агента на суттєво більшій кількості епох, створення масштабнішої нейронної мережі на вході та розширення множини дій та станів для забезпечення можливості підтримки більш цікавих сценаріїв.

ПЕРЕЛІК ПОСИЛАНЬ

1. Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness (Дата звернення: 10.12.2020).

2. Goodfellow, I., Bengio, Y., Courville, A. Deep learning. Massachusetts : MIT press, 2017. 511 p.

3. Minsky M. Neural Nets and the Brain Model Problem Princeton: Princeton University, 1954. 157 p.

4. Tesauro G. Practical Issues in Temporal Difference Learning, *Machine Learning*, 1992, vol. 8. P. 257–277.

5. Кузнєцова Н. В. Порівняльний аналіз характеристик моделей оцінювання ризиків кредитування / ред. Н. В. Кузнєцова, П. І. Бідюк / *Наукові вісті НТУУ «КПІ»*, 2010. No1. С. 42–53.

6. Архітектура нейронних мереж. – Режим доступу: http://techn.sstu.ru/kafedri/подразделения/1/MetMat/Terin/neiro/neiro.html (Дата звернення: 10.12.2020)

7. Моделі нейронних мереж. – URL: https://www.intuit.ru/studies/courses/57/57/lecture/1682?page=3 (Дата звернення: 10.12.2020)

8. Bain A. The Senses and the Intellect. London: Parker, 1855. 215 p.

9. Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 1958. Vol 65, No 6. P. 386-408.

10. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems,* 2017. Vol. 28, No 10. P. 2222-2232.

11. Richard S. Sutton, Andrew G. Barto Reinforcement Learning, second edition. An Introduction. London: The MIT Press, 2018. 526 p.

12. Micheal Lanham, Hands-On Reinforcement Learning for Games. Birmingham: Packt Publishing, 2020. 432 p.

13. Huang, Te-Ming, Kecman, Vojislav, Kopriva Ivica Kernel Based Algorithms for Mining Huge Data Sets, in Supervised, Semi-supervised, and Unsupervised Learning. Berlin: Springer-Verlag, 2006. 260 p.

14. Howard R. A. Dynamic Programming and Markov Processes, Cambridge, MA: MIT Press, 1960. 231 p.

15. Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller Playing Atari with Deep Reinforcement Learning URL: - https://arxiv.org/abs/1312.5602 (Дата звернення: 10.12.2020).

16. Vincent François-Lavet, Peter Henderson, Riashat Islam Marc G. Bellemare, Joelle Pineau An Introduction to Deep Reinforcement Learning. [S.l.]: Now Foundations and Trends, 2018. 156 p.

17. Marko Lukša Kubernetes in Action. Shelter Island: Manning, 2017. 624 p.

18. Jeff Nickoloff and Stephen Kuenzli Docker in action. Shelter Island: Manning, 2016. 336 p.

19. Laura Graesser, Wah Loon Keng Foundations of Deep Reinforcement Learning. Boston: Addison-Wesley Professional, 2019. 379 p.

20. Mataric, M. J. Reward functions for accelerated learning. *Lecture Notes in Computer Science,* 1994. Vol. 4131. P. 181-189.

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

```
import numpy as np
import matplotlib.pyplot as plt
import pickle

plt_legend_dict = {"expected_sarsa_agent": "Expected SARSA with neural network",
                   "random_agent": "Random",
                   "requests_submitted": "requests_submitted",
                   "node_counts": "node_counts",
                   "requests_in_progress": "requests_in_progress",
                   "pending_requests": "pending_requests",
                   "total_latency": "total_latency"
                   }
path_dict = {"expected_sarsa_agent": "results/",
             "random_agent": "./"}

plt_label_dict = {"expected_sarsa_agent": "Sum of\nreward\nduring\nepisode"}

def smooth(data, k):
    num_episodes = data.shape[1]
    num_runs = data.shape[0]

    smoothed_data = np.zeros((num_runs, num_episodes))

    for i in range(num_episodes):
        if i < k:
            smoothed_data[:, i] = np.mean(data[:, :i+1], axis = 1)
        else:
            smoothed_data[:, i] = np.mean(data[:, i-k:i+1], axis = 1)


    return smoothed_data

# Function to plot result
def plot_result(data_name_array):
    plt_agent_sweeps = []

    fig, ax = plt.subplots(figsize=(8,6))


    for data_name in data_name_array:
```

```
            # load data
            filename = 'sum_reward_{}'.format(data_name).replace('.','')
            sum_reward_data = np.load('{}/{}.npy'.format(path_dict[data_name], filename))

            # smooth data
            smoothed_sum_reward = smooth(data = sum_reward_data, k = 100)

            mean_smoothed_sum_reward = np.mean(smoothed_sum_reward, axis = 0)

            plot_x_range = np.arange(0, mean_smoothed_sum_reward.shape[0])
                    graph_current_agent_sum_reward, = ax.plot(plot_x_range, mean_smoothed_sum_reward[:],
label=plt_legend_dict[data_name])
            plt_agent_sweeps.append(graph_current_agent_sum_reward)

        ax.legend(handles=plt_agent_sweeps, fontsize = 13)
        ax.set_title("Learning Curve", fontsize = 15)
        ax.set_xlabel('Episodes', fontsize = 14)
        ax.set_ylabel(plt_label_dict[data_name_array[0]], rotation=0, labelpad=40, fontsize = 14)
        ax.set_ylim([-300, 300])

        plt.tight_layout()
        plt.show()


    def plot_demo(title, data_dict, ylim = [-150, 150]):
        # node_counts, requests_in_progress, pending_requests, total_latency
        plt_agent_sweeps = []

        fig, ax = plt.subplots(figsize=(8, 6))

        for data_name, data_array in data_dict.items():
            # load data
            # filename = 'sum_reward_{}'.format(data_name).replace('.', '')
            # data = data_dict[data_name]
            #
            # # smooth data
            # smoothed_sum_reward = smooth(data=sum_reward_data, k=100)
            #
            # mean_smoothed_sum_reward = np.mean(smoothed_sum_reward, axis=0)

            plot_x_range = np.arange(0, data_array.shape[0])
            graph_current_agent_sum_reward, = ax.plot(plot_x_range, data_array[:],
                                                      label=plt_legend_dict[data_name])
            plt_agent_sweeps.append(graph_current_agent_sum_reward)
```

```
    ax.legend(handles=plt_agent_sweeps, fontsize=13)
    ax.set_title(title, fontsize=15)
    ax.set_xlabel('Time step', fontsize=14)
    # ax.set_ylabel(plt_label_dict[data_name_array[0]], rotation=0, labelpad=40, fontsize=14)
    ax.set_ylim(ylim)

    plt.tight_layout()
    plt.show()
```

```
#!/usr/bin/env python

"""Glues together an experiment, agent, and environment.
"""

from __future__ import print_function
import numpy as np
from plot_script import plot_demo


class RLGlue:
    """RLGlue class

    args:
        env_name (string): the name of the module where the Environment class can be found
        agent_name (string): the name of the module where the Agent class can be found
    """

    def __init__(self, env_class, agent_class):
        self.environment = env_class()
        self.agent = agent_class()

        self.total_reward = None
        self.last_action = None
        self.num_steps = None
        self.num_episodes = None

    def rl_init(self, agent_init_info={}, env_init_info={}):
        """Initial method called when RLGlue experiment is created"""
        self.environment.env_init(env_init_info)
        self.agent.agent_init(agent_init_info)

        self.total_reward = 0.0
        self.num_steps = 0
        self.num_episodes = 0
```

```
def rl_start(self, agent_start_info={}, env_start_info={}):
    """Starts RLGlue experiment

    Returns:
        tuple: (state, action)
    """

    self.total_reward = 0.0
    self.num_steps = 1

    last_state = self.environment.env_start()
    self.last_action = self.agent.agent_start(last_state)

    observation = (last_state, self.last_action)

    return observation

def rl_agent_start(self, observation):
    """Starts the agent.

    Args:
        observation: The first observation from the environment

    Returns:
        The action taken by the agent.
    """
    return self.agent.agent_start(observation)

def rl_agent_step(self, reward, observation):
    """Step taken by the agent

    Args:
        reward (float): the last reward the agent received for taking the
            last action.
        observation : the state observation the agent receives from the
            environment.

    Returns:
        The action taken by the agent.
    """
    return self.agent.agent_step(reward, observation)

def rl_agent_end(self, reward):
    """Run when the agent terminates
```

```
    Args:
        reward (float): the reward the agent received when terminating
    """
    self.agent.agent_end(reward)

def rl_env_start(self):
    """Starts RL-Glue environment.

    Returns:
        (float, state, Boolean): reward, state observation, boolean
            indicating termination
    """
    self.total_reward = 0.0
    self.num_steps = 1

    this_observation = self.environment.env_start()

    return this_observation

def rl_env_step(self, action):
    """Step taken by the environment based on action from agent

    Args:
        action: Action taken by agent.

    Returns:
        (float, state, Boolean): reward, state observation, boolean
            indicating termination.
    """
    ro = self.environment.env_step(action)
    (this_reward, _, terminal, _) = ro

    self.total_reward += this_reward

    if terminal:
        self.num_episodes += 1
    else:
        self.num_steps += 1

    return ro

def rl_step(self):
    """Step taken by RLGlue, takes environment step and either step or
        end by agent.
```

```
    Returns:
        (float, state, action, Boolean): reward, last state observation,
            last action, boolean indicating termination
    """

    (reward, last_state, term, info) = self.environment.env_step(self.last_action)

    self.total_reward += reward

    if term:
        self.num_episodes += 1
        self.agent.agent_end(reward)
        roat = (reward, last_state, self.last_action, term, info)
    else:
        self.num_steps += 1
        self.last_action = self.agent.agent_step(reward, last_state)
        roat = (reward, last_state, self.last_action, term, info)

    return roat

def rl_cleanup(self):
    """Cleanup done at end of experiment."""
    self.environment.env_cleanup()
    self.agent.agent_cleanup()

def rl_agent_message(self, message):
    """Message passed to communicate with agent during experiment

    Args:
        message: the message (or question) to send to the agent

    Returns:
        The message back (or answer) from the agent

    """

    return self.agent.agent_message(message)

def rl_env_message(self, message):
    """Message passed to communicate with environment during experiment

    Args:
        message: the message (or question) to send to the environment
```

```
    Returns:
        The message back (or answer) from the environment

    """
    return self.environment.env_message(message)

def rl_episode(self, max_steps_this_episode, start=True):
    """Runs an RLGlue episode

    Args:
        max_steps_this_episode (Int): the maximum steps for the experiment to run in an episode

    Returns:
        Boolean: if the episode should terminate
    """
    is_terminal = False
    last_action = 0

    if start:
        self.rl_start()

    while (not is_terminal) and ((max_steps_this_episode == 0) or
                           (self.num_steps < max_steps_this_episode)):
        rl_step_result = self.rl_step()
        is_terminal = rl_step_result[3]
        last_action = rl_step_result[2]

    return is_terminal, last_action

def rl_episode_demo(self, max_steps_this_episode, start=True):
    """Runs an RLGlue episode

    Args:
        max_steps_this_episode (Int): the maximum steps for the experiment to run in an episode

    Returns:
        Boolean: if the episode should terminate
    """
    is_terminal = False
    last_action = 0
    if start:
        self.rl_start()

    agent_actions = np.zeros(max_steps_this_episode)
```

```
        node_counts_0 = np.zeros(max_steps_this_episode)
        node_counts_1 = np.zeros(max_steps_this_episode)
        node_counts_2 = np.zeros(max_steps_this_episode)
        node_counts_3 = np.zeros(max_steps_this_episode)

        requests_submitted = np.zeros(max_steps_this_episode)
        pending_requests = np.zeros(max_steps_this_episode)
        total_latency = np.zeros(max_steps_this_episode)

        current_node_count_0 = 0
        current_node_count_1 = 0
        current_node_count_2 = 0
        current_node_count_3 = 0
        requests_submitted_count = 0
        step = 0
        while (not is_terminal) and ((max_steps_this_episode == 0) or
                                     (self.num_steps < max_steps_this_episode)):
          (reward, last_state, last_action, term, info) = self.rl_step()
          is_terminal = term

          if last_action == 0:
            current_node_count_0 = current_node_count_0 + 1
          if last_action == 1:
            current_node_count_1 = current_node_count_1 + 1
          if last_action == 2:
            current_node_count_2 = current_node_count_2 + 1
          if last_action == 3:
            current_node_count_3 = current_node_count_3 + 1

          requests_submitted_count = requests_submitted_count + info["ox"]
          pending_requests[step] = info["ox"]
          requests_submitted[step] = requests_submitted_count



          agent_actions[step] = last_action
          node_counts_0[step] = current_node_count_0
          node_counts_1[step] = current_node_count_1
            node_counts_2[step] = np.max([0, current_node_count_2 - current_node_count_3 - pending_re-
quests[step]])
          node_counts_3[step] = current_node_count_3
          step = step + 1

        # plot_demo("node_counts_0", {
        #     "node_counts": node_counts_0
```

```
    # }, [-50, 50])

    # plot_demo("node_counts_1", {
    #     "node_counts": node_counts_1
    # }, [-50, 50])
    # plot_demo("node_counts_2", {
    #     "node_counts": node_counts_2
    # }, [-50, 50])
    # plot_demo("node_counts_3", {
    #     "node_counts": node_counts_3
    # }, [-50, 50])
    # plot_demo("requests_in_progress", {
    #     "requests_in_progress": requests_in_progress
    # }, [-50, 50])
    plot_demo("", {
        # "total_latency": total_latency,
        "requests_submitted": requests_submitted,
        "node_counts": node_counts_2,
        "requests_in_progress": node_counts_3
    }, [-50, 50])
    # plot_demo("total_latency", {
    #     "total_latency": total_latency
    # }, [-50, 50])
    # plot_demo("agent_actions", {
    #     "total_latency": agent_actions
    # }, [0, 4])

def rl_return(self):
    """The total reward

    Returns:
        float: the total reward
    """
    return self.total_reward

def rl_num_steps(self):
    """The total number of steps taken

    Returns:
        Int: the total number of steps taken
    """
    return self.num_steps

def rl_num_episodes(self):
    """The number of episodes
```

```
        Returns
            Int: the total number of episodes

        """
        return self.num_episodes

#!/usr/bin/env python

"""RandomWalk environment class for RL-Glue-py.
"""

from environment import BaseEnvironment
import numpy as np
import gym
from agent_base import OrchtopusBase

class OrchtopusEnvironment(BaseEnvironment):
    def env_init(self, env_info={}):
        """
        Setup for the environment called when the experiment first starts.
        """
        self.env = OrchtopusBase()
        self.env.seed(0)

    def env_start(self):
        """
        The first method called when the experiment starts, called before the
        agent starts.

        Returns:
            The first state observation from the environment.
        """

        reward = 0.0
        observation = self.env.reset()
        is_terminal = False

        self.reward_obs_term = (reward, observation, is_terminal)

        # return first state observation from the environment
        return self.reward_obs_term[1]

    def env_step(self, action):
        """A step taken by the environment.
```

```
Args:
    action: The action taken by the agent

Returns:
    (float, state, Boolean): a tuple of the reward, state observation,
        and boolean indicating if it's terminal.
"""

last_state = self.reward_obs_term[1]
current_state, reward, is_terminal, info = self.env.step(action)

self.reward_obs_term = (reward, current_state, is_terminal, info)

return self.reward_obs_term
```