МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Навчально-науковий інститут новітніх освітніх технологій
Кафедра комп'ютерних наук

ЛУКАЧАТ Андрій Олегович

Математичне та програмне забезпечення для управління IT-інфраструктурою / Mathematical Tools and Software for IT Infrastructure Management

спеціальність: 121 - Інженерія програмного забезпечення
освітньо-професійна програма - Інженерія програмного забезпечення

Кваліфікаційна робота

Виконав студент групи ІПЗзм-21
А. О. Лукачат

Науковий керівник:
к.т.н., доцент С. Я. Крепич

Кваліфікаційну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
А. В. Пукас

ТЕРНОПІЛЬ - 2022

ЗМІСТ

ВСТУП

Актуальність теми. Хмарні обчислення стали широко поширеною технологією надання доступу до обчислювальних ресурсів. Головним чином, це обумовлено тим, що хмарні системи можуть забезпечити практично будьякий вид обслуговування і звільнити клієнтів від необхідності створення фізичних інфраструктур.

У зв’язку зі збільшенням попиту на хмарні сервіси, оптимізація використання ресурсів хмарних інфраструктур стає ще більш важливою, оскільки її вирішення дасть змогу підвищити продуктивність, доступність та надійність таких систем. Технології віртуалізації виявились досить зручними для вирішення таких проблем. Віртуалізація серверів дозволяє ефективно розподіляти фізичні обчислювальні ресурси між додатками та користувачами: декілька віртуальних машин можуть працювати ефективно та ізольовано, незалежно від їх фізичного розміщення. Завдяки цьому, провайдери хмарних сервісів мають можливість обслуговувати більшу кількість клієнтів гнучким та ефективним способом.

Зв'язок роботи з науковими програмами, планами, темами

Напрямок виконаних досліджень безпосередньо пов'язаний з науково-дослідним напрямком кафедри “комп’ютерних наук” Західноукраїнського національного університету.

Мета і задачі дослідження

Метою роботи є підвищення ефективності використання ресурсів програмних проектів, які використовують елементи хмарних IT-інфраструктур.

Виходячи із поставленої мети, у магістерському дослідженні вирішуються такі завдання:

- аналіз проблем розподілу критичних ресурсів програмних проектів;
- аналіз існуючих методів, математичних моделей та алгоритмів;
- розробка методів розподілу критичних ресурсів з використанням технологій штучного інтелекту, зокрема генетичних алгоритмів;
- проведення експериментальних досліджень застосування розроблених методів;
- розробка інструментальних засобів управління ресурсами програмних проектів.

Об'єкт дослідження – процеси управління ресурсами програмних проектів.

Предмет дослідження – методи та програмні засоби оцінки ресурсів програмних проектів.

Методи дослідження. Для розв'язання поставлених задач використано: при розробленні методу, моделей, засобів і алгоритмів – теорія системного аналізу, методи штучного інтелекту; при розробленні математичних моделей – теорія математичного моделювання; при розробленні програмних моделей – принципи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів. вдосконалено генетичний алгоритм в сфері розв'язання проблем управління ресурсами в частині знаходження функції пристосованості кожного індивіду, де для кожного індивіда у популяції вибирається пара для генерації індивідів наступної популяції.

Практичне значення одержаних результатів. Для практичного використання отриманих теоретичних результатів розроблено програмне забезпечення, що може використовуватися для управління ресурсами програмних проектів.

Особистий внесок магістранта. Всі результати отримані автором самостійно.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Аналіз проблем розподілу ресурсів в програмних проектах

В умовах глобальної інформатизації суспільства і бізнесу, перед кожною організацією виникає дилема – створювати, розвивати і підтримувати власні програмні проекти, або користуватися послугами компаній, які надають необхідні послуги з інформаційного і програмно-апаратного забезпечення реалізації автоматизованих бізнес-процесів компанії-користувача. Якщо мова йде про велику кількість користувачів і потужні розподілені проекти, дилема переформулюється таким чином – створювати, розвивати і підтримувати власні центри обробки даних, чи користуватися послугами хостингових компаній, які надають хостингові послуги користувачам системи створених і підтримуваних ними центрів.

Якщо компанія зупиняється на першій альтернативі, то, цілком природно, їй необхідно забезпечити ефективне функціонування своїх проектів. Важливо, що результатом цього звичайно є зменшення витрат на експлуатацію, яке супроводжується зниженням цін для користувачів і дозволить, зрештою, закласти основу для ефективного бізнесу компаній, які пішли іншим шляхом [3].

Склалося загальне уявлення про центри опрацювання даних як комплексного організаційно-технічного рішення, призначеного для створення високопродуктивної, відмовостійкої інфраструктури. Сучасні центри орієнтовані на вирішення бізнес завдань шляхом надання послуг у вигляді інформаційних сервісів. До основних завдань центрів в першу чергу

відносяться ефективне консолідоване зберігання і оброблення даних користувачів, надання їм прикладних сервісів, а також підтримка функціонування корпоративних програмних додатків.

Започаткована порівняно недавно концепція уже втілена багатьма великими корпораціями переважно для забезпечення доступу великої кількості користувачів до певних ресурсів (сервісів, застосувань, обчислювальних потужностей, даних тощо). В Україні ця концепція перспективна, в першу чергу, для міністерств і відомств, яким притаманна наявність розподіленої системи підпорядкованих підприємств, організацій, установ і окремих підрозділів. Якщо врахувати, що центральні органи міністерства (відомства), як і підпорядковані підприємства, організації, установи і окремі підрозділи, мають необхідність підтримувати певні інформаційно-обчислювальні потужності, то виникає традиційна проблема пошуку різноманітних форм збільшення ефективності їх використання. Схожі проблеми виникнуть і у будь-якої компанії, яка прийме рішення вкласти кошти у створення системи для обслуговування користувачів інших компаній, тобто набуде статусу хостингової компанії.

Останнім часом дуже часто можна зустріти надання сервісів кінцевим користувачам через так званні хмари, або хмарні сервіси. Існує декілька моделей надання хмарних послуг. NIST (National institute of Standards and Technology, U.S. Department of Commerce) в своїй спеціальній публікації 800-146 [4] виділив три моделі хмарних сервісів (рис. 1.1):

- IaaS (Infrastructure as a Service);
- PaaS (Platform as a Service);
- SaaS (Software as a Service).

Перший шар «Infrastructure as a Service», або IaaS. Іноді його ще називають «Hardware as a Service», або HaaS. Ще кілька років тому, якщо хтось хотів запускати бізнес-застосування у своєму офісі і керувати веб-сайтом компанії, він мав придбати сервери та інше дороге обладнання, щоб

управляти локальними програмами та забезпечити безперебійне ведення бізнесу.

Але тепер, з IaaS, є можливість орендувати апаратні ресурси у компанії, які надають такі послуги. На відміну від оренди звичайної виділених серверів або оренди віртуальних приватних серверів, коли замовник сплачує орендну платню навіть у випадку простоїв обладнання і лише звільняється від витрат на обслуговування техніки і виділення офісних приміщень під сервери, надання сервісів IaaS у хмарах, може оплачуватись по факту використання обчислювальних потужностей. Найбільшими гравцями в цьому секторі є Amazon, Microsoft, VMWare, Rackspace, Red Hat.

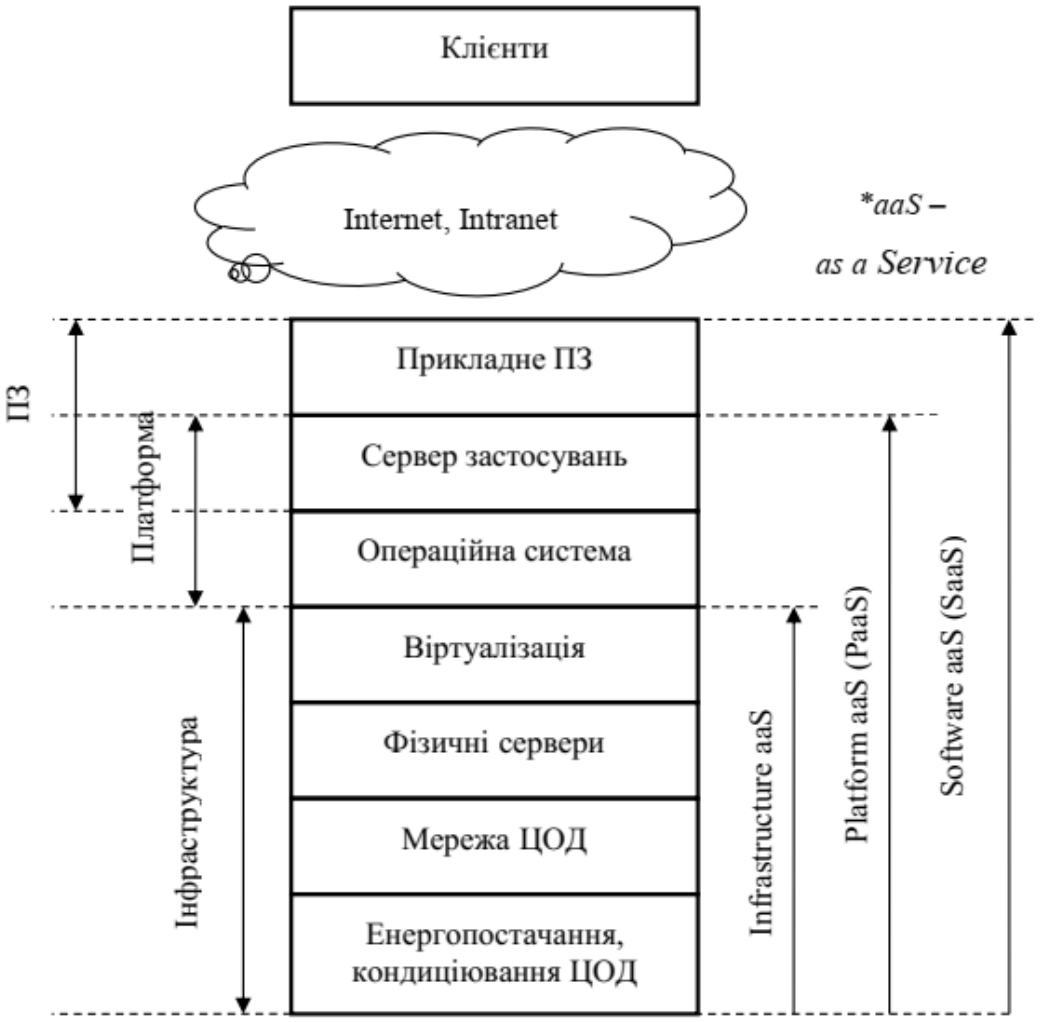

Рис. 1.1. Модель надання хмарних сервісів

Другий шар хмарних сервісів відомий як «Platform as a Service», або PaaS. Основна ідея цієї категорії полягає в тому, що вся розробка власного програмного забезпечення (ПЗ) компанії може здійснюватись в цьому шарі, заощаджуючи час і ресурси за рахунок відсутності у необхідності підтримки middleware за власний кошт.

PaaS-компанії пропонують широкий спектр рішень для розробки і розгортання застосувань через Інтернет. Це дозволяє заощадити гроші на обладнанні, а також спрощує спільну роботу для географічно-розподіленої команди розробників ПЗ.

Найбільш відомими сервісами такого типу є Google App Engine, Microsoft Azure, Saleforce's Force.com, the Salesforce-owned Heroku, Engine Yard.

Третій і останній шар хмар – «Software as a Service», або SaaS. Саме з цим шаром частіше всього працюють кінцеві користувачі IT-послуг у повсякденному житті. Будь-яке застосування, розміщене на віддаленому сервері, яке може бути доступне через Інтернет вважається SaaS. Послуги, які кінцевий користувач споживає повністю з Інтернету, такі як Netflix, MOG, Google Apps, Box.net, Dropbox, iCloud від Apple належать до цієї категорії.

1.2. Побудова програмних проектів та їх вплив на задачі управління ресурсами

Комплекс задач дослідження й їх постановки залежать від багатьох чинників, які тією чи іншою мірою впливають на згадані вище процеси. Доцільно перейти до виявлення цих чинників, оцінювання їх впливу та особливостей його врахування при розв'язанні зазначених проблем. Розпочнемо з аналізу моделей хостингу. Кожна з хостингових компаній

зазвичай надає послуги на основі таких чотирьох моделей хостингу для клієнтів (останні, при переростанні можливостей однієї моделі мають можливість перейти на іншу, з як правило більш дорогим сервісом, або миритись з обмеженням можливостей):

- віртуальний хостинг – це хостинг для великої кількості користувачів, сотні і тисячі акаунтів яких можуть розміщуватись на одному сервері, змагаючись за серверні ресурси. Так, веб-сайти на сервері розділяють серверні ресурси, включаючи дисковий простір, пропускну здатність мережевих інтерфейсів, процесорний час. Звичайно цю технологію використовують для клієнтів з не дуже серйозними вимогами;
- віртуальні приватні сервери – це технологія спільного використання ресурсів, коли клієнти не змагаються за ресурси, а мають можливість орендувати ресурси на сервері і ці ресурси резервуються для них. Така технологія називається віртуалізацією;
- виділений сервер – деякі користувачі обирають цю технологію, вважаючи, що тільки це захистить їх від перевантаження серверів, поганої продуктивності та недосконалих програм їх сусідів по хостингу;
- колокейшн – це спеціальний вид послуг, який може надавати хостингова організація. Полягає він в тому, що сервери, які належать клієнтові розміщуються на майданчику хостингової компанії. Термін колокейшн походить від англійського слова *colocation*, що означає «спільне розміщення».

Далі необхідно зважити на роль технологій. Постійне зростання потреб клієнтів зажадало від хостингових організацій впровадження ряду технічних рішень, які дозволяють масштабувати системи, гнучко розподіляти ресурси між всіма клієнтами і балансувати навантаження, що поступає.

Архітектура визначає чим є власне система і чим вона не є. Вона повинна будуватися на абстракціях, правилах і обмеженнях, які визначають, що може робити система.

Сьогодні технологій, які використовують хостингові організації вистачає багато, але практично всі користуються такими технологіями:

- трирівнева (n-tier) архітектура;
- віртуалізація;

Класична клієнт-серверна архітектура передбачає наявність двох програм. Перша – сервер баз даних – стежить за збереженням даних, розміщує їх на зовнішньому носії та виконує запити, що приходять, на пошук і оновлення даних. Друга програма – програма-клієнт – забезпечує інтерфейсну взаємодію з користувачем, формує запити до сервера бази даних, посилає їх та отримує потрібні дані, які показує користувачеві. Програма-клієнт і програма-сервер можуть фізично знаходитися як на одному комп'ютері, так і на різних.

Використання архітектури клієнт-сервер дозволило прискорити доступ до даних (сервер не займається інтерфейсом і логікою застосувань), будувати гетерогенні мережі (клієнт і сервер спілкуються по мережевому протоколу, і можуть функціонувати на абсолютно різних платформах).

Останнім часом став виникати синдром "товстого клієнта". Це означає, що клієнтське застосування має розмір, порівнянний або такий, що навіть перевищує розмір програми-сервера бази даних. Таке застосування вимагає від комп'ютераклієнта дуже великих ресурсів. А клієнтських місць в реальній інформаційній системі, як правило, досить багато. Тому загальні витрати на устаткування (пам'ять, диски) для комп'ютерів-клієнтів можуть бути невиправдано великими. З іншого боку, програми-клієнти в одній інформаційній системі з великою вірогідністю містять шматки кодів, що повторюються. Наприклад, якщо використовувана інформаційна система враховує податки, то вона повинна рахувати суму податку на всіх клієнтах однаково. Таким чином, один і той же код багато разів продубльований і на кожному з комп'ютерів-клієнтів займає деякі, вельми відчутні ресурси.

При деяких видах обробки інформації, що зберігається в базі даних, потрібно передавати по мережі великі об'єми інформації. Наприклад, в базі даних зберігається інформація про мільйон платіжних доручень. Урядом було прийнято постанову про зміну податків. Треба перерахувати податок на кожен документ. Якщо це реалізувати на клієнтові, то тоді по мережі потрібно буде передати практично всі дані і виграшу архітектура клієнт-сервер не дасть.

Рішенням для даної проблеми могло бути проведення обробки безпосередньо на тому ж комп'ютері, де зберігаються дані. Сервери баз даних Oracle підтримують механізм збережених процедур – підпрограм, що мають доступ до системи керування базою даних. Для програмування збережених процедур використовується спеціальна мова – PL/SQL. У багатьох випадках простіше б було розробляти алгоритм обробки на тій же мові, в тій же системі програмування, що і всю клієнтську частину, але виконувати цей алгоритм не на машині-клієнтові, а на машині, де знаходиться сервер баз даних.

У деяких системах розробки застосувань є засоби створення сервера застосувань. За допомогою цих бібліотек можна розбивати своє клієнтське застосування на дві частини. Одна частина буде традиційним клієнтом і знаходиться на комп'ютері-клієнті, а друга частина, сервер-застосувань, знаходиться на іншому комп'ютері, можливо якраз на тому, де розташований сервер бази даних. Таким чином, використовуючи сервер застосувань можна пом'якшити або звести нанівець недоліки традиційної архітектури «клієнт-сервер».

Ідея сервера застосувань полягає в розбитті застосування на дві частини: власне клієнта і сервера даного застосування. Причому сервер застосувань може бути один на багато застосувань. Клієнти спілкуються з сервером застосувань. Клієнти посилають серверу застосувань запити, а

отримують відповіді. Клієнти можуть звернутися і безпосередньо до сервера бази даних за тими або іншими даними.

Звернення за даними до сервера бази даних може проводити і сервер застосувань.

Додавання третього рівня забезпечує ряд принципових переваг:

- Сегментація. Переміщенням частини коду на вторинні сервери можна сегментувати велику частину логіки і збільшити кількість контрольних точок.
- Розподіл процесорного навантаження. Інтерфейсні Web-сервера в цій моделі в основному займаються наданням статичного вмісту, зверненнями до об'єктів другого рівня і здобуттям відповідей.
- Надмірність. Можливість розміщення серверів застосувань на кластерах підвищує надійність системи.
- Абстрагування бази даних. Відділення рівня бізнес-логіки від рівнів баз даних та інтерфейсу, спрощує вирішення проблем масштабування.
- Оптимізація серверів. Розділення баз даних, Web-серверів і бізнес-логіки дозволяє оптимізувати кожен сервер і забезпечити найкращу швидкодію при виконанні конкретного круга завдань. Крім того, скорочення функціональності серверів наводить до зменшення непродуктивних витрат.

Додавання нового рівня вимагає додаткової роботи і додаткового планування.

Створення надійного скомпільованого коду додасть роботи програмістам. Опрацювання зв'язків між рівнями також означає додаткове архітектурне планерування. Крім того, ускладнюються середовища розробки і публікації програмного коду. Хоча поява нового рівня відкриває нові можливості для оптимізації, оцінка навантажень і процес ухвалення рішень також стають складнішими. Але в перспективі ці витрати значно окупаються за рахунок підвищення надійності, масштабованості та спрощення підтримки таких застосувань.

Сьогодні практично будь-яке правильно побудоване веб-сервером застосування є трирівневим, що дозволяє зручним чином для хостингових компаній розподілити навантаження і ресурси.

Ідеологія побудови йде від абстракції сервісів. Її мета полягає у створенні на прийнятному рівні абстракції сервісів, розрахованих на багаторазове використання в багатьох застосуваннях, а не жорстко прив’язаних до одного застосування [5].

OGSA підтримує створення та обслуговування ансамблів сервісів, які використовуються у віртуальних організаціях. Тут сервіс визначений як об'єкт із підтримкою мережевої роботи, що забезпечує деякі такі можливості, як обчислювальні ресурси, ресурси пам'яті, мережі, програми та бази даних. Це пристосовує створення Web-сервісів для виконання деяких вимог, специфічних для Grid. Нижче наведені стандартні інтерфейси, наявні в OGSA [7]:

- Виявлення (discovery): клієнти потребують механізми для того, щоб знайти доступні сервіси та визначити особливості цих сервісів і мати можливість їх конфігурувати відповідно своїх запитів до цих сервісів.
- Динамічне створення сервісу (Dynamic service creation): стандартний інтерфейс і семантика, що повинні забезпечити створення будь-якого сервісу. - Довічне керування (Lifetime management): у системі повинні бути надані механізми для відновлення сервісів і його стану, пов'язані з невдалими операціями.
- Повідомлення (Notification): безліч динамічних, розподілених сервісів повинні бути здатними виявити один одного та одержати інформацію про зміну їхнього стану.
- Керованість (Manageability): надані операції, що відносяться до керування та контролю великої кількості екземплярів класів Grid-сервісів.
- Просте хост-середовище (Simple hosting environment): просте виконавче середовище – це ряд ресурсів, розташованих у межах окремого

адміністративного домена, що підтримує рідні засоби для керування сервісом: наприклад, сервер додатків J2EE, система Microsoft.NET, або кластер Linux.

Web-сервіси забезпечують функціональну сумісність, тобто ключ до Grid обчислень, а OGSA впроваджує засоби адаптації Web-сервіси до Grid. Однак, Webсервіси не забезпечують нове рішення багатьох із проблем масштабних розподілених систем з надвеликою кількістю запитів, і при цьому вони також не забезпечують нові методики для розробки цих систем. Отже, для обслуговування широкого кола запитів використовуються інші моделі, наприклад, засновані на програмних агентах [8]. При цьому агенти мають наступні особливості:

- превентивність – агенти демонструють поведінку, що спрямована на рішення поставленої мети;
- автономія – агенти діють без зовнішнього втручання та мають деякий контроль над своїми діями та внутрішнім станом;
- соціальна здатність – агенти взаємодіють із іншими агентами, використовуючи мову комунікації агента;
- реактивність – агенти зберігають і відповідають за своє середовище.

Обчислення, засноване на агентах, особливо добре підходить для динамічно змінного середовища, де автономія агентів дозволяє адаптувати обчислення до змінних обставин.

Поняття віртуалізації умовно можна розділити на дві категорії, що фундаментально розрізняються: - віртуалізація платформ; - віртуалізація ресурсів.

Під віртуалізацією платформ розуміють створення програмних систем на основі існуючих апаратно-програмних комплексів, залежних або незалежних від них. Система, що надає апаратні ресурси і програмне забезпечення, називається хостовою (host), а симулюючі системи –

гостьовими (guest). Аби гостьові системи могли стабільно функціонувати на платформі хостової системи, необхідно, щоб програмне і апаратне забезпечення хоста було досить надійним і надавало необхідний набір інтерфейсів для доступу до його ресурсів. Є декілька видів віртуалізації платформ, в кожному з яких здійснюється свій підхід до поняття «віртуалізації». Види віртуалізації платформ залежать від того, наскільки повно здійснюється симуляція апаратного забезпечення. Віртуалізація платформ розглядає поняття віртуалізації у вузькому сенсі, переважно застосовуючи його до процесу створення віртуальних машин (ВМ) – програмних абстракцій, що запускаються на платформі реальних апаратно-програмних систем.

Віртуалізація ресурсів розглядає віртуалізацію в широкому сенсі, узагальнюючи підходи до створення ВМ і переносячи їх на усі види ресурсів з метою комбінування та спрощення представлення апаратних ресурсів для користувача і формування деяких призначених для користувача абстракцій обладнання інформаційно-телекомунікаційних систем, просторів імен, мереж і т.п. Цей вид віртуалізації дозволяє концентрувати, абстрагувати і спрощувати управління групами ресурсів, таких як мережі, сховища даних і простору імен тощо [9].

Використання програмної віртуалізації дозволяє реалізувати наступні переваги:

- підвищення норми утилізації ресурсів з 5-15 % до 60-80 % приводить до економії на придбанні серверів, оренді площ, зниженні енергоспоживання серверів і систем кондиціювання, зниження потреби в обслуговуючому персоналі;
- розгортання віртуальних серверів і застосувань та їхнє перенесення на інше обладнання займає набагато менше часу;
- відкривається можливість динамічно виділяти ресурси кожної ВМ відповідно до реальних потреб застосувань;

- кожна з ВМ повністю ізольована від інших ВМ. Крах будь-якої ВМ не впливає на працездатність інших ВМ, що функціонують на тій же апаратурі;
- централізовані засоби управління всіма віртуальними серверами підприємства через єдиний інтерфейс;
- висока оперативність відновлення доступності застосувань після збоїв.

Щоб віртуальні системи чи ресурси могли стабільно функціонувати на платформі хостової системи чи реальних ресурсів, необхідно розв'язати низку проблем, аналогічних наведеним вище, але з урахуванням особливостей технологій віртуалізації. Програмне і апаратне забезпечення хостових систем повинне бути досить надійним і надавати необхідний набір інтерфейсів для доступу до їх ресурсів. Крім того, має бути створене ПЗ ефективного управління ресурсами хостових систем з урахуванням, з одного боку можливостей і стану хостових систем, з іншого боку, потреб віртуальних систем та рівня і прогнозних оцінок їх завантаженості. Виявляється необхідність у розробленні відповідних моделей, алгоритмів і засобів моніторингу, аналізу, планування і управління ресурсами ЦОД з урахуванням специфіки технологій віртуалізації.

Основною властивістю кластерів є висока готовність застосувань і досягається вона завдяки закладеній у кластери надмірності складових частин. Більшість кластерних рішень оснащена механізмами розпізнавання відмов вузла і забезпечує передачу завдань вузла, що відмовив, іншому. Крім того, кластерні файлові системи пропонують деякі функції, що гарантують високу готовність. Зазвичай вони реалізуються як свого роду «оболонки» регулярної файлової системи, взаємодіють між собою по мережі для синхронізації дій і використовують ту ж мережу зберігання даних (Storage Area Network, SAN) або спеціальні пристрої. Типові функції – механізм ведення журналів, де фіксуються всі дії, і менеджер блокувань, керівник

доступом до даних у всій системі. Найбільш відомими кластерними файловими системами є глобальна файлова система (Global File System, GFS) у варіанті з відкритими початковими кодами OPENGFS і використовувана компанією Red Hat, що в свій час була придбана у компанії Sistina комерційна Pendant (також GFS). Крім того, на ринку представлені узагальнена паралельна файлова система (Generalized Parallel File System, GPFS) від IBM, кластерна файлова система (Cluster File System, CFS) від Polyserve і Sun Cluster.

Майже кожен виробник апаратного забезпечення пропонує власне готове рішення для забезпечення критеріїв готовності, балансування навантажень, паралельної обробки даних, шифрування та забезпечення безпеки при передачі та обробці результатів обчислень. Проаналізуємо ці готові рішення.

Технологія кластерного хостингу розроблена для обмеження проблем, які присутні в типовій розділеній інфраструктурі хостингу. Ця технологія дозволяє користувачам керувати безпекою, балансуванням навантажень і ресурсами вебсайтів. Одразу після замовлення акаунта користувач може використовувати такі ресурси хостингу, як DNS-сервіси, email, FTP, системи керування базами даних (СКБД) і т.п. Кластерний хостинг дозволяє користувачам замовляти додатковий дисковий простір, трафік, процесорний час та інші ресурси, які не є фізичним обмеженням платформи.

Кластерний хостинг агрегує ресурси, які надаються лімітованими фізичними серверами. Клієнти не обмежуються одним сервером. Вони використовують процесорний час декількох серверів і їх навантаження розподіляється в реальному часі. Це означає, що вони можуть замовити стільки серверних ресурсів, скільки вони бажають отримати від віртуальних серверів. Слід зазначити, що навіть найбільші клієнти потребуватимуть тільки частину серверного пулу. Зміни в акаунті клієнта

(для додавання нових ресурсів або зміни налаштувань) поширюються негайно на кожен сервер в кластері. Це відмінність від типових архітектур хостингу спільного використання, які зазвичай потребують, щоб зміни в конфігураційному файлі набрали чинності після перезавантаження сервера або застосовуються на циклічній основі кожних декілька годин.

Трафік динамічно балансується по декількох вузлах, що зводить до мінімуму вплив зростаючих навантажень. Збільшення обчислювальних ресурсів може бути реалізовано шляхом додавання додаткових серверів в кластер, що з економічної точки зору є значно більш ефективним, ніж заміна існуючих серверів. Це може бути здійсненим дуже швидко і без негативного впливу на існуючих користувачів. Оскільки користувач обслуговуються декількома реальними серверами, а не одним, то використання серверних ресурсів сусідом не вплине на час відповіді веб-сайту або на доступні йому ресурси.

1.3. Аналіз відомих хмарних технологій

Виконаємо порівняння моделей даних хмарних технологій. Дані для порівняння наведені в таблиці 1.1.

Таблиця 1.1.

Порівняльна характеристика моделей хмарних обчислень з різними правами доступа клієнта до обчислювальних ресурсів.

	Власний сервер	Інфраструктура (як сервіс)	Платформа (як сервіс)	Програмне забезпечення (як сервіс)
Застосування	+	+	+	-
Дані	+	+	+	-
Час виконання	+	+	-	-
Проміжний шар	+	+	-	-
О/С	+	+	-	-
Віртуалізація	+	-	-	-

Amazon Web Service має досить зручний інтерфейс для контролю за станом хмари, кількістю розгорнутих віртуальних машин, логічних дисків і т.п. Приклад інтерфейсу Amazon Web Service наведено на рисунку 1.3.

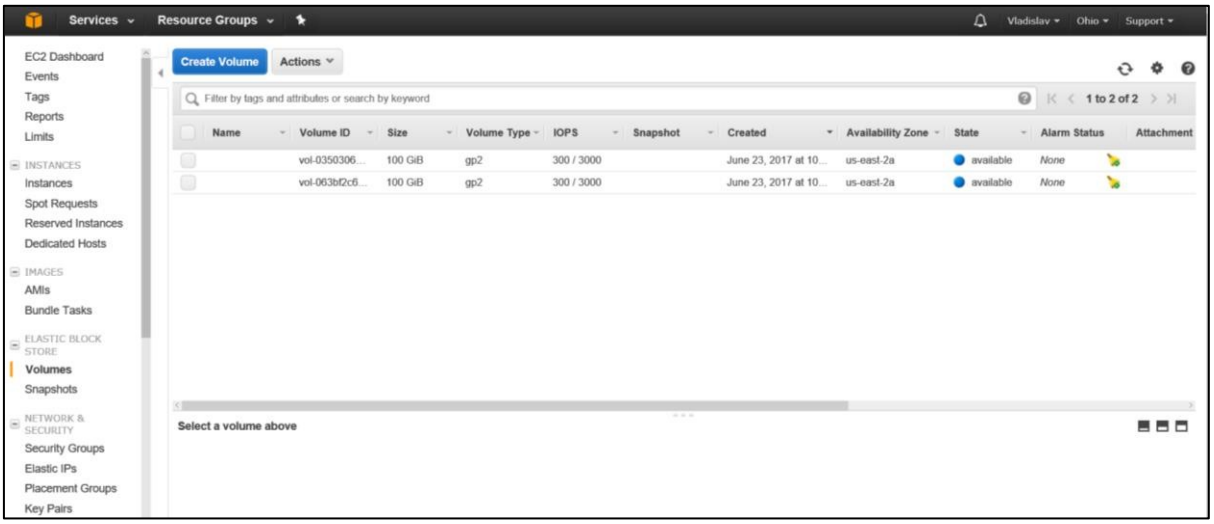

Рис. 1.3. Інтерфейс користувача в Amazon Web Service

Asure наведено на рисунку 1.5.

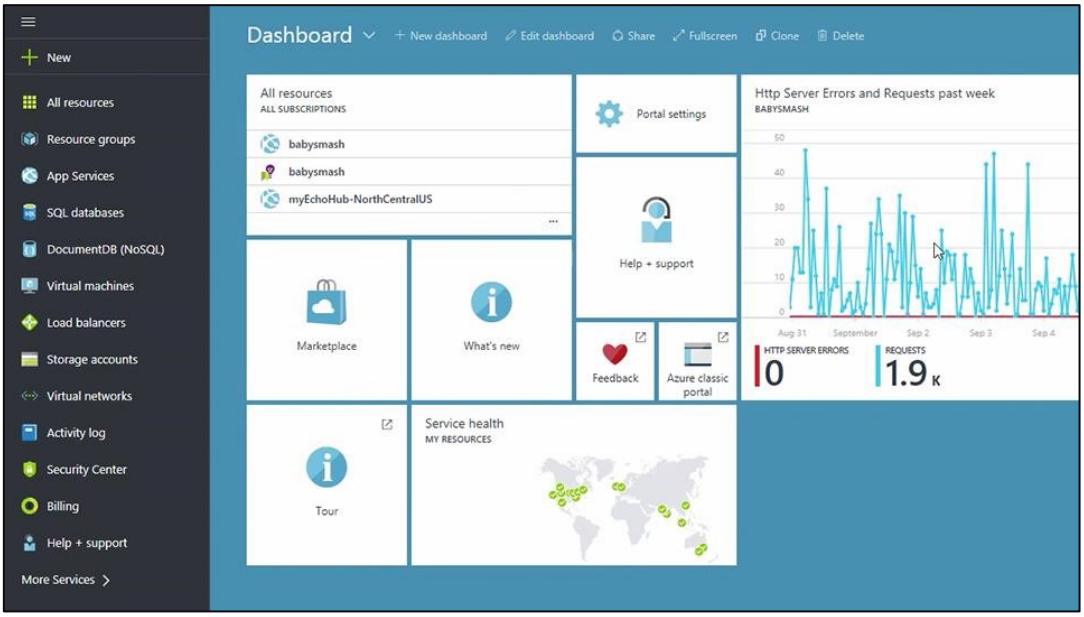

Рис. 1.5. Інтерфейс користувача в Microsoft Azure

Результати порівняння провайдерів наведені в таблиці 1.2. Нині на ринку хмарних обчислень ці три постачальники послуг є найбільшими конкурентами, хоча Amazon Web Services має значну фору. Тому оберемо цього провайдера хмарних обчислень для подальшо використання під даний проект.

Таблиця 1.2

Порівняльна характеристика найпопулярніших провайдерів IaaS

	Amazon Web Services (AWS)	Microsoft Azure	Google Compute Engine
Кількість сімейств віртуальних машин	7	4	4
Кількість типів віртуальних машин	38	33	18
Наявність регіонів	+	+	+
Наявність зон	+	-	+
Ціноутворення	За годину, округлюється.	За хвилину, округюється (з передплатою або щомісяця)	За хвилину, округлюється (мінімум 10 хв)
Моделі оплати	На вимогу, зарезервовано за місцем.	За запитом – короткострокові зобов'язання (оплачені або щомісяця)	На вимогу – стале використання
Віртуальна мережа	VPC	VNet	Підмережа
Публічний IP	+	+	+
Гібридні хмари	+	+	-
DNS	Маршру- тизується	-	-
Firewall/ACL	+	+	+
Зберігання об’єктів	S3	Block Blobs and Files	Google Cloud Storage
Блок зберігання	EBS	Page Blobs	Стійкі диски

Amazon EC2 – веб-сервіс, котрий надає користувачу обчислювальні потужності в хмарі. Даний сервіс входить в інфраструктуру Amazon Web Services.

Цей сервіс має простий веб-інтерфейс, що дозволяє отримати доступ до обчислювальних потужностей та налаштувати ресурси з мінімальними

затратами за короткий час. Користувач має повний контроль за обчислювальними ресурсами, а також доступне середовище для роботи.

За допомогою EC2 користувач може:

- створювати Amazon Machine Image (AMI), котрий буде утримувати програми, бібліотеки, дані та пов'язані з ними функціональні параметри або використовувати раніше налаштовані шаблони образів та робіт;
- завантажити AMI в Amazon S3. Amazon EC2 пропонує інструменти, для зберігання AMI. Amazon S3 забезпечує захищене, надійне та швидке сховище для зберігання образів;
- використовувати Amazon EC2 веб-сервіс для налаштування безпеки та мережевого доступу;
- обирати тип операційної системи, за власними потребами, запускати, вимикати або контролювати декілька AMI по мірі необхідності, використовувати API веб-сервісу, або різних інструментів управління, які раніше були передбачені;
- визначити необхідність локалізації в декількох місцях, використовуючи статичні IP-адреси або інші варіанти;

оплачувати тільки за ресурси, котрі будуть використовуватись, такі як час або кількість переданої інформації (кількість трафіку).

1.4. Аналіз відомих моделей оцінки ресурсів програмних проектів

На рисунку 1.6 представлено модель розподілу обмежених ресурсів.

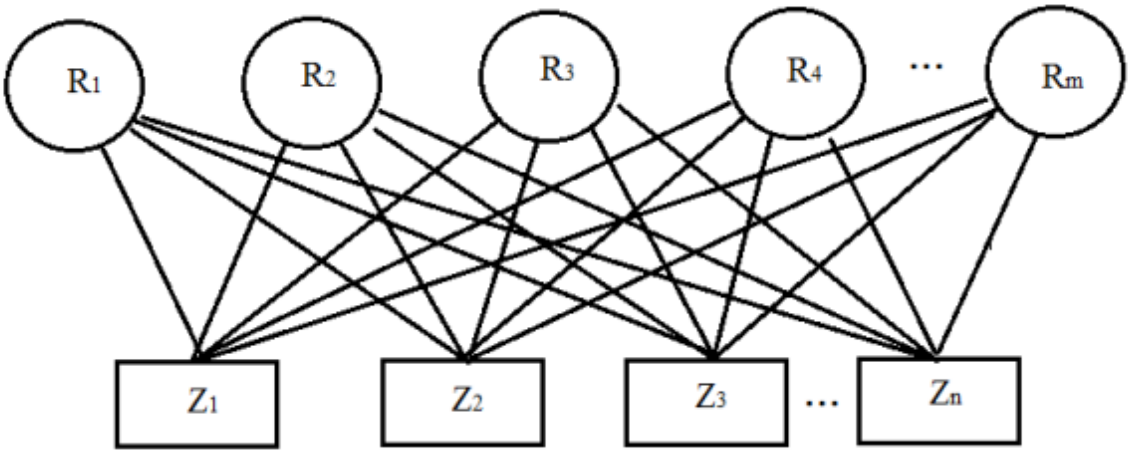

Рис. 1.6. Модель розподілу обмежених ресурсів

Введемо необхідні для побудови моделей наступні позначення:

- $Z_{1,\dots,}Z_n$ – бізнес-процеси, підтримку яких забезпечує функціонування проекту;
- $W = (w_1, \dots, w_n)$ – коефіцієнти важливості бізнес процесів $Z_{1,\dots,}Z_n$ відповідно;
- $R_{1,\dots,}R_m$ – інтегровані ресурси проекту, необхідні для підтримки функціонування бізнес-процесів;
- $P = \|\rho_{ij}\|$– матриця потреб бізнес-процесів у ресурсах проекту, де ρ_{ij} відповідає наведеній кількості потрібного для процесу Z_i ресурсу R_j чи 0, якщо ресурс не потрібен;
- $D = \|d_{ij}\|$– – матриця наявності потреб бізнес-процесів у ресурсха ІТС, де:

$$\begin{cases} d_{ij} = 1, \text{ якщо } \rho_{ij} > 0 \\ d_{ij} = 1, \text{ якщо } \rho_{ij} = 0 \end{cases}$$

- R = $(r_1, \dots, r_m)$ – вектор встановлених обмежень на ресурси.

Розглянемо дискретний випадок, коли бізнес-процес або підтримується в повному обсязі, або повністю блокується. Введемо вектор

X = $(x_1, \dots, x_n)$, де:

$$x_i = \begin{cases} 1, & \text{якщо процес } Z_i \text{ обслуговується} \\ 0, & \text{в протилежному випадку} \end{cases}$$

Тоді критерій оптимального розподілу ресурсів можна подати у вигляді:

$$max \sum_{i=1}^{n} w_i \, x_i$$

Мають також бути використані обмеження по ресурсам

$$\sum_{i=1}^{n} x_i \, p_{ij} \leq r_j, j = 1, \dots m$$

Висновки до першого розділу

1. У розділі було виконано аналіз IT інфраструктури як об’єкта управління, визначено набір її основних характеристик. Виконано аналіз існуючих проблем в управлінні IT інфраструктурою, встановлено можливі способи їх вирішення. Зазначено важливість проблеми створення інструментарію для технічної підтримки IT інфраструктури.

2. Розглянуто декілька існуючих технологій управління IT інфраструктурою. Враховуючи їх переваги та недоліки, було обгрунтовано доцільність створення власної технології управління ресурсами. Були визначено головні її функціональні та технічні вимоги.

РОЗДІЛ 2
МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ОЦІНКИ РЕСУРСІВ В ПРОГРАМНИХ ПРОЕКТАХ

2.1. Методи розподілу ресурсів у віртуазілованому середовищі

Які наслідки несе консолідація віртуальних машин на один фізичний сервер? В яких ситуаціях це може мати переваги? На перший погляд, філософії віртуалізації та високої доступності протиречать одна одній. Отже, як правильно вибрати фізичну платформу для запезпечення високої доступності у віртуалізованих середовищах? Іноді у випадках, коли фізичний сервер демонструє надзвичайну стійкість на безперебійність роботи, доцільно розміщувати різні компоненти критичної інфраструктури на одному фізичному сервері. Недоліком таких систем є висока вартість: високодоступні платформи корпоративного рівня включають значні об’єми надлишкових ресурсів на різних рівнях системи, різні джерела живлення та численні способи мережевого підключення.

Інший підхід заключається у виборі декількох фізичних серверів для розміщення компонентів критичної інфраструктури. Цей підхід частіше використовується у системах некорпоративного рівня для компенсування витрат на апаратні засоби, хоча значна частина корпоративних систем також використовує його.

У односерверному високодоступному середовищі компоненти критичної інфраструктури можуть скористатись усіма можливостями віртуалізації та розподілу фізичних ресурсів. Оскільки увесь зв’язок між компонентами критичної інфраструктури здійснюється в межах одного фізичного серверу, зазвичай такі системи демонструють знижені викладні витрати на шифрування трафіку у внутрішній мережі. Також відсутність

великого об’єму мережевої взаємодії приводить до збільшення відклику системи загалом.

Крім того, такі системи є вигіднішими, ніж рішення, що вимагають декількох копій одного і того ж фізичного обладнання, з точки зору вартості придбання та подальших витрат на управління та експлуатацію. Існують також і інші фінансові вигоди, які можуть бути не настільки очевидні. На деяких платформах, наприклад "IBM System Z", додатки можуть бути встановлені на кожному екземплярі без будьяких додаткових витрат на програмне забезпечення.

Хоча розгортання компонентів критичної інфраструктури на одному фізичному сервері вводить єдину точку відмови, ця проблема може бути частково вирішена придбанням більш надійного обладнання. Проте, в такій ситуації простої при запланованому обслуговуванні в значній мірі неминучі. Щоб мінімізувати проблему єдиної точки відмови, віртуалізовані конфігурації повинні включати в себе запуск декількох гостьових операційних систем, розгортання надлишкових віртуальних мереж, забезпечення надлишкових підключень до зовнішньої інфраструктури, залучення віртуалізованих розподілених систем зберігання даних. При можливості, кілька програмних гіпервізорів повинні бути розгорнуті у різних логічних розділах для ще більшого резервування. У порівнянні із багато-серверними архітектурами, цей підхід є відносно вигіднішим.

Для усунення потенційних планових та позапланових простоїв необхідно здійснити розширення віртуалізованого односерверного середовища. При розподіленні компонентів критичної інфраструктури необхідно пересвідчитись в тому, що ні один кластер не розміщує усі вузли на одному фізичному сервері.

Розглянемо конфігурацію, що зображена на рисунку 2.2, що включає у себе 2 фізичних вузла, кожен з яких знаходиться під управлінням гіпервізора, що підтримує дві гостьові системи в режимі "Active/Active".

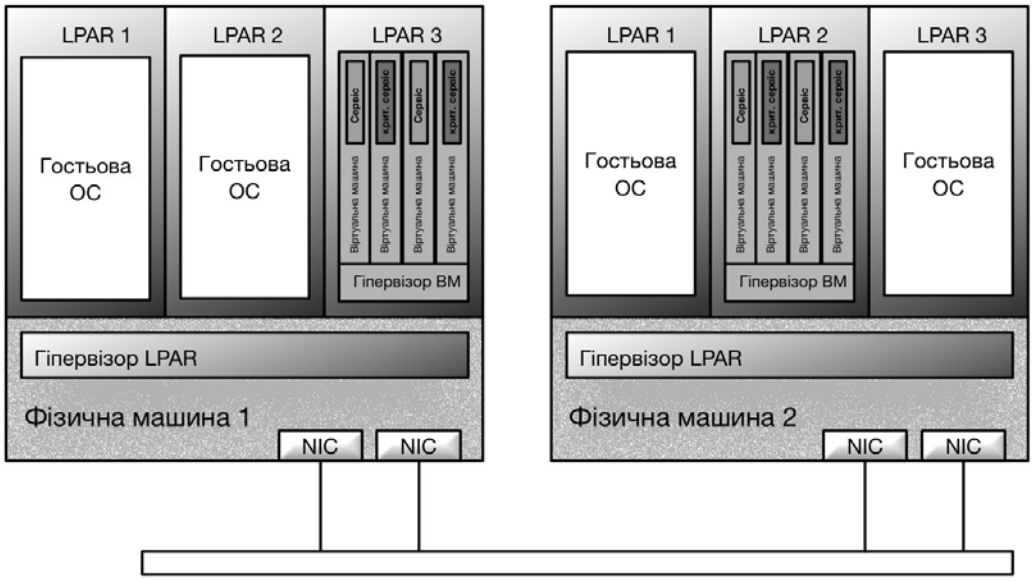

Рис. 2.2. Віртуалізоване середовище що охоплює декілька хостів

Нехай деяка критична задача розподілена між 4-ма віртуальними машинами, що знаходяться на різних дискретних фізичних системах. У такому випадку відмова одного гіпервізору або серверу загалом не призводить до краху системи.

Розподіл фізичних ресурсів у межах одного хосту забезпечує значну ефективність їх використання для найбільш ймовірних сценаріїв відмови, так що об’єми надлишкових резервних ресурсів є мінімальними. Часто гіпервізор може бути налаштований таким чином, що навіть у випадку відмови хосту відбудеться автоматичний перерозподіл системних ресурсів для використання робочими вузлами, що знову ж таки зменшує об’єми необхідних резервних ресурсів. Проте економія ресурсів є незначною у порівнянні із односерверними розміщеннями. Багатосерверні конфігурації несуть у собі додаткові приховані апаратні витрати, такі як збільшення складності мереж, застосування криптографічних засобів для взаємодії між вузлами, збільшення мережевої затримки.

Останнім цікавим аспектом багатосерверних віртуалізованих середовищ є використання технологій міграцій віртуальних машин, таких як "VMware VMotion", "IBM z/VM Cross System Extensions (CSE)", "XenSource

XenMotion" та інші. Ці технології дозволяють здійснювати переміщення віртуальних машин від одного хосту до іншого, в деяких випадках без здійснення перерви в обслуговуванні. Оскільки ця функціональність є досить нова, існує великий інтерес у цій галузі. Хоча технології міграцій віртуальних машин можуть забезпечити високу доступність інфраструктури при запланованих або позапланових простоях, поки не ясно як ця функціональність вплине на вибір розгортання при виборі конфігурацій "Active/Active" або "Active/Passive".

Хоча багатосерверні розгортання усувають єдині точки відмови, забезпечують більшу гнучкість запланованих простоїв та надають простий спосіб масштабування , шляхом додавання нових вузлів, існують деякі проблеми при їх використанні. Віртуальні високодоступні середовища, що з самого початку були розроблені з ціллю охоплення декількох фізичних хостів, дозволять подолати ці проблеми, інкрементально додавати інші фізичні та відповідні віртуальні хости при необхідності. Збільшення вартості обладнання та складності системи дозволить запланованим/позаплановим простоям не впливати на загальну доступність системи.

2.2. Розподіл ресурсів із врахуванням вимог доступності

Досі ми розглядали методи, що розраховують оптимальне розміщення компонентів критичної інфраструктури на етапі запуску інфраструктури. Проте під час експлуатації можуть виникати ситуації, що потребують корегувати поточне розміщення:

- збільшення споживання ресурсів компонентом інфраструктури;
- відмова одного або декількох серверів;

- необхідність планового оновлення обладнання або програмного забезпечення.

У системах з підвищеними вимогами до надійності повинні існувати засоби динамічного балансування навантаження. Для здійснення цієї вимоги необхідно модифікувати фітнес-функцію таким чином, щоб вона враховувала властивості як початкового розміщення, так і нового. Розроблена фітнес-функція представлена у (2.7) та складається із трьох компонентів. На відміну від (2.7), в якій компоненти марнування та перевикористання ресурсів поєднувались через множення безрозмірних величин, у (2.7) усі компоненти приведені до однієї величини – фінансових витрат, що понесе компанія-власник інфраструктури. Така реалізація функції оцінки є простішою, більш наявною та краще піддається аналізу.

$$F(A_0,A) = F_w(A) + F_o(A) + F_m(A_0, A) \quad (2.7)$$

де A_0 - стан початкового розміщення; A - стан нового розміщення; $F_w(A)$ - витрати марнування ресурсів інфраструктури за нового розміщення; $F_o(A)$ – витрати, пов’язані із недостатнім забезпеченням сервісів інфраструктури; $F_m(A_0, A)$ - витрати на приведення інфраструктури із стану A_0 до стану A.

Таким чином, мінімальне значення фітнес-функції буде являти собою оптимальне співвідношення між марнуванням ресурсів та забезпеченням сервісів. Розглянемо кожен її компонент окремо.

Для оцінки витрат, пов’язаних з марнуванням ресурсів інфраструктури (2.8), для кожного ресурсу необхідно додатково задати величину середніх витрат одиниці ресурсу, що несе власник інфраструктури. Для прикладу, для розрахунку таких витрат можна використати амортизаційну вартість обладнання, що підтримує ресурс у наявності, об’єм електроенергії, що споживається обладнанням або об’єм упущеної вигоди, що могла б бути отримана. Очевидно, що коли певний ресурс використовується в повному або надмірному обсязі, це значення дорівнює нулю.

Для гарантії забезпечення сервісів необхідним об’ємом ресурсів, компонент фітнес-функції (2.8) визначає штраф, що стягується при порушенні такої умови. Очевидно, що коли сервіс отримує необхідний об’єм ресурсів, це значення дорівнює нулю. Величина штрафу є нелінійною та залежить від об’єму фактично наданих ресурсів. Приклад зображений на рис. 2.2.

$$F_o(A) = \sum_{s \in S} \sum_{r \in R} Y_s^r$$

$$Y_s^r = \begin{cases} 0, & \text{якщо } v_{rec}^r \geq v_{req}^r \\ K_s \left(\dfrac{v_{req}^r}{v_{rec}^r} - 1 \right), & \text{інакше} \end{cases}$$

де $v_{req}{}^r$ - об’єм ресурсу, що вимагає сервіс для своєї роботи;

$v_{rec}{}^r$ - об’єм наданого ресурсу для використання сервісом;

K_s - штраф внаслідок падіння швидкодії сервісу, що зумовлена двократною нестачею ресурсу.

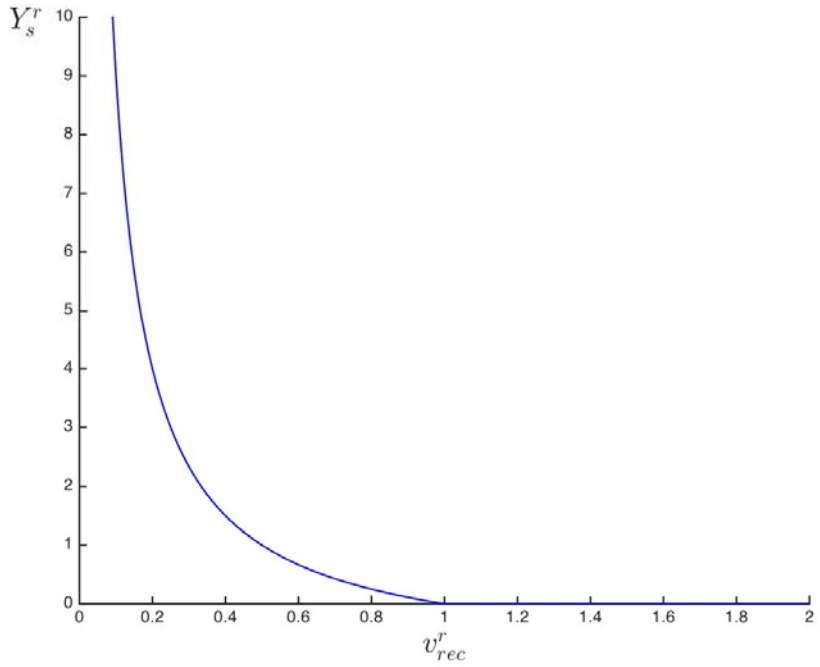

Рис. 2.2. Залежність величини штрафу від об’єму наданих ресурсів при $K_s = 1$, $v_{req}{}^r = 1$

Останній компонент фітнес-функції відповідає за мінімізацію витрат, пов’язаних із приведенням системи з одного стану до іншого: міграцією сервісів. Для урахування вартості здійснених міграцій необхідно визначити функцію вартості переміщення сервісу, що оцінює складність міграції сервісу з початкового стану у кінцевий. Важливим елементом будь якої сучасної технічної системи є спеціалізоване програмне забезпечення.

Висновки до другого розділу

1. Розглядаючи проблему розподілу ресурсами як задачу багатомірного пакування, було розроблено кілька способів застосування технологій штучного інтелекту для її вирішення. Розроблені моделі призначені для досягнення максимальної утилізації обчислювальних ресурсів, водночас мінімізуючи кількість застосованих фізичних машин.

2. Застосування розроблених моделей для розподілу елементів критичної IT інфраструктури передбачає врахування вимог високої доступності. У розділі було продемонстровано метод, що дозволяє забезпечити виконання цих вимог, шляхом введення додаткових ресурсів.

РОЗДІЛ 3
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ОЦІНКИ РЕСУРСІВ В ПРОГРАМНИХ ПРОЕКТАХ

3.1. Архітектура та функціональність системи

Для розробки програмного забезпечення керування компонентами критичної інфраструктури була обрана мова Python та фреймворк Django. Вибір мови Python обумовлений тим, що дана мова програмування є дуже зручною для створення різноманітних прототипів та моделей, а також існує велика кількість бібліотек для задач лінійного програмування, швидких обчислень, забезпечення прямого доступу до хмарних технологій з використанням певного клієнту або алгоритмів штучного інтелекту. Також варто зазначити великі можливості Python щодо зручного представлення даних.

Динамічна типізація та зручний синтаксис, велика кількість вбудованих структур даних, а також простота створення нових структур даних – все це робить мову Python дуже зручною для швидкого прототипування, реалізації різних математичних моделей. Для мови Python було обрано середовище розробки PyCharm. PyCharm розроблений компанією JetBrains на основі IntelliJ IDEA. PyCharm являє собою інтегроване середовище розробки, що працює під операційними системам Windows, Mac OS X та Linux.

Система має архітектуру типу “клієнт-сервер”. Діаграма розгортання системи наведена на рисунку 3.1. Загалом, система складається з двох частин: серверний додаток, що виконує навчання нейронної мережі та реалізує алгоритм структурної оптимізації; клієнтський додаток, що реалізує графічний інтерфейс користувача.

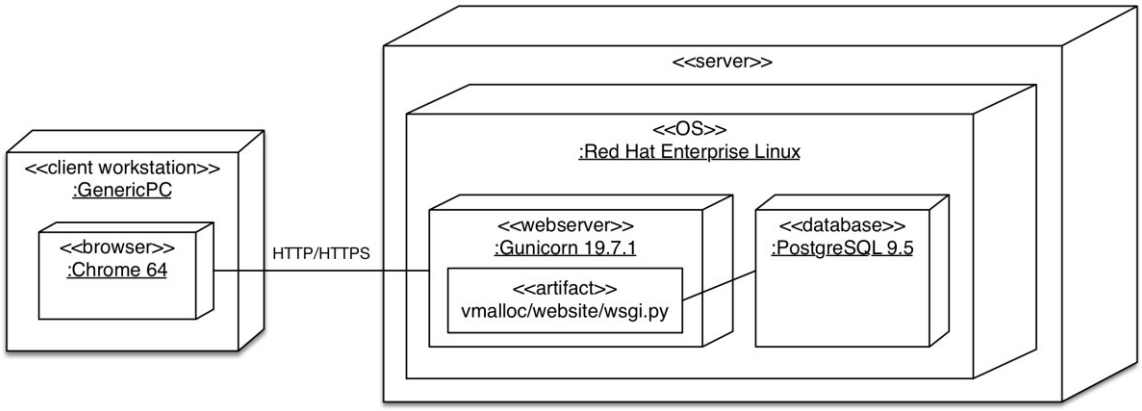

Рис. 3.1. Діаграма розгортання системи

Логічно система розділена на 4 компоненти, кожен з яких може бути замінений незалежно один від одного (рис. 3.2). До складу входять:

Головний модуль управління – здійснює аналіз, планування та управління станом IT-інфрастурктури;

Система розрахунку розміщення – здійснює пошук субоптимального розміщення елементів IT-інфраструктури за прийнятний час.

Модуль рішення задачі пакування – здійснює пошук субоптимального рішення задачі багатомірного пакування за прийнятних час. Модуль є необхідним для розрахунку розміщення;

Система управління інфраструктурою – здійснює фактичне управління станом об’єктів IT інфраструктури.

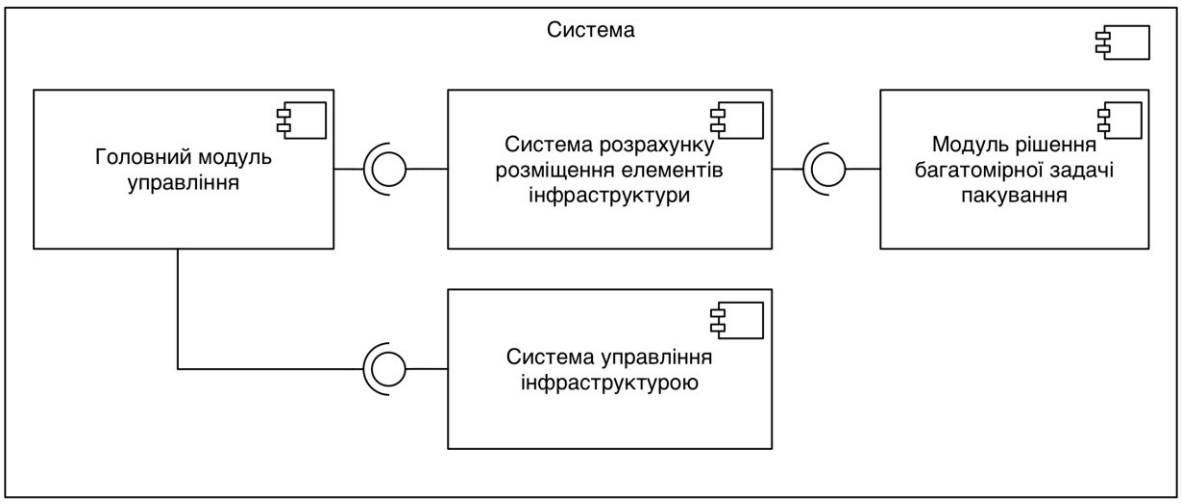

Рис. 3.2. Діаграма компонентів системи

Взаємодія із системою здійснюється за допомогою веб-інтерфейсу, у якому множина доступних операцій визначається в залежності від ролі користувача:

- роль «клієнт» - користувач, що є власником критичних сервісів, що підтримуються за рахунок потужностей IT-інфраструктури;

- роль «оператор» - володіє адміністративними правами та є представником компанії-власника IT-інфраструктури.

Залежності між ролями користувачів та доступними операціями зображені у вигляді діаграми прецедентів на рисунку 3.3.

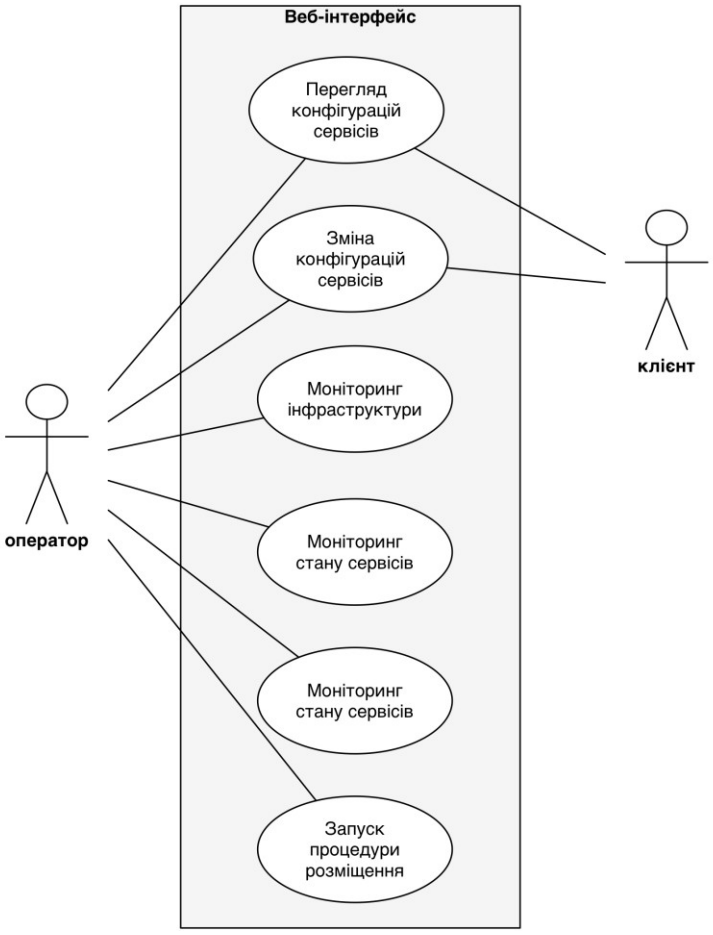

Рис. 3.3. Діаграма варіантів використання

Також набір доступних операцій залежить від стану, в якому перебуває система. На рис. 3.4 зображена діаграма станів системи. У стані “узгоджено” усі сервіси розміщені таким чином, що вимоги кожного сервісу повністю задовільняються. При зміні вимог хоча б одного сервісу або при зміні

конфігурації інфраструктури система переходить у стан “Оцінка”, в якому оператор здійснює якісну оцінку необхідності застосування нового розміщення. При підтвердженні оператором система переходить у стан “Розміщення”, в якому відбувається запуск, зупинка або фізичне переміщення компонентів інфраструктури. Після завершення усіх операцій, що приводять компоненти інфраструктури до нового стану, система переходить у стан “узгоджено” та процес повторюється знову.

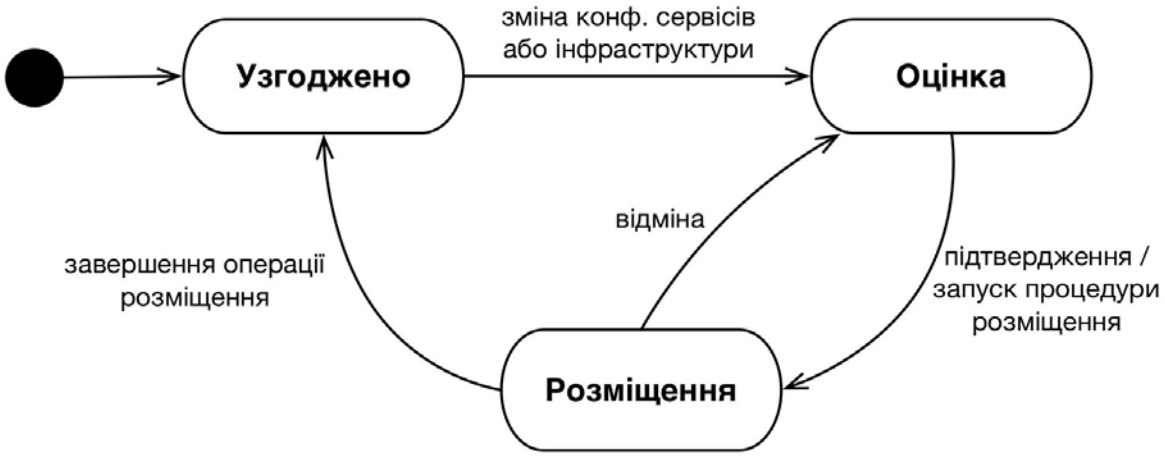

Рис. 3.4. Діаграма станів системи

Компоненти системи виконані у вигляді окремих Python-пакетів. У контексті моделі трирівневої архітектури структура пакетів та їх залежності зображені на рисунку 3.5:

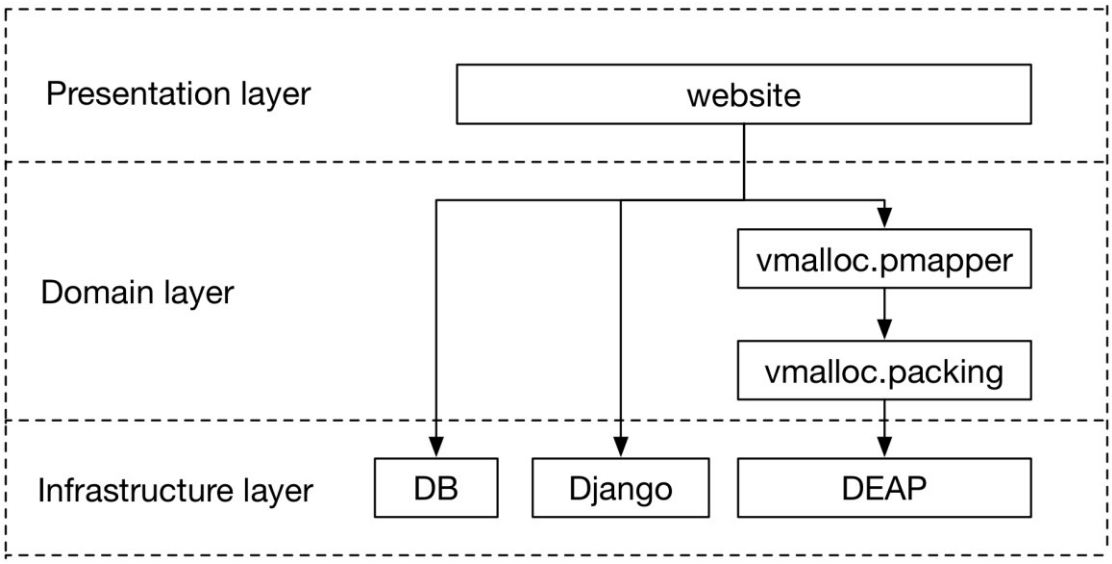

Рис. 3.5. Реалізація моделі трирівневої архітектури

До складу програмного забезпечення входять наступні Python-пакети:

vmalloc – ядро системи, що розраховує оптимальне розміщення віртуальних машин та обчислювальних процесів критичної інфраструктури. Пакет надає інтерфейс у вигляді чистих функцій, що приймають та повертають незмінні структури данних (namedtuple). Не виконує будь-якого вигляду роботу із базами даних, зовнішніми системами або іншими statefull об’єктами;

vmalloc.packing – універсальний модуль, що реалізує вирішення задачі багатомірного пакування. До складу програмного інтерфейсу входять структури даних *Ресурс* (Resource), *Предмет* (Item), *Контейнер* (Container) та *Середовище* (Environment). Бібліотека, відповідно, розраховує оптимальне розміщення предметів серед контейнерів, що є найбільш оптимальним з точки зору використання ресурсів. Оскільки цей модуль є досить абстрактним (буквально нічого не знає про фізичні та віртуальні машини), використовувати його надзвичайно легко і для вирішення інших комбінаторних задач. Реалізує алгоритми генетичного програмування, метод рою часток та інші. Використовує потужний фреймворк DEAP для реалізації еволюційних алгоритмів;

vmalloc.pmapper (Process Mapper) – головний модуль, що відповідає за розрахунок розміщення компонентів критичної інфраструктури. До складу програмного інтерфейсу входять структури даних для представлення ресурсів, фізичних та віртуальних машин, ЦОД та програмних процесів, що виконуються у них. Містить одну функцію vmalloc.pmapper.allocate, що для наданої конфігурації системи (ресурси, процеси, ЦОД) повертає об’єкт *Розміщення* (Placement), що описує оптимальне розміщення віртуальних машин серед елементів ЦОД, та оптимальне розміщення програмних процесів серед віртуальних машин. На обох рівнях використовує модуль рішення задачі багатомірного пакування. Діаграма класів модуля наведена на рисунку 3.6.

website – веб-інтерфейс платформи управління критичною інфраструктурою, що реалізований за допомогою фреймворку Django. Містить опис моделей бази даних, реалізацію інтерактивного веб-інтерфейсу та синхронізацію результатів, отриманих за допомогою vmalloc.pmapper та базою даних.

website.exim – реалізовує завантаження та збереження конфігурацій систем у форматі JSON.

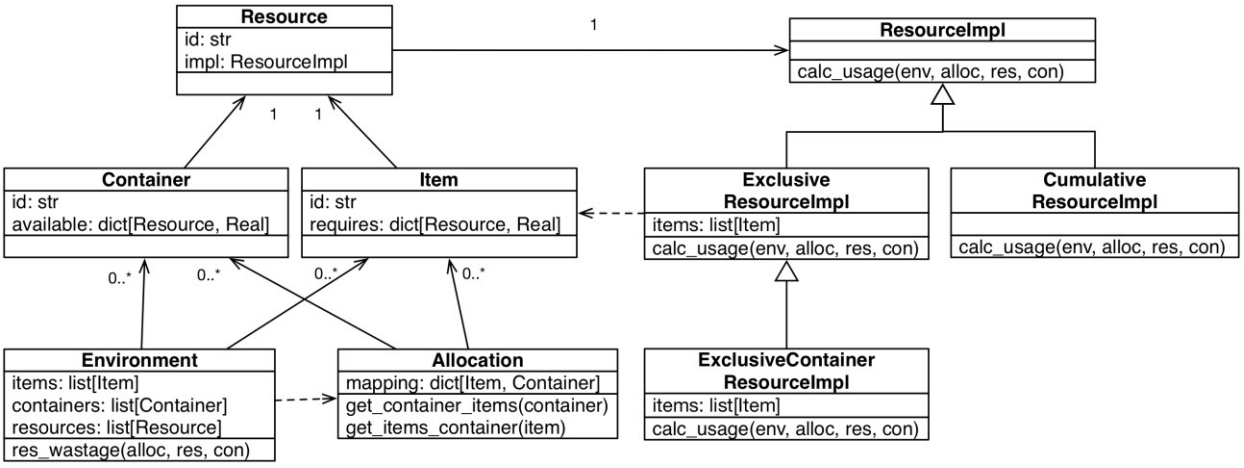

Рис. 3.6. Діаграма класів пакету vmalloc.pmapper

3.2. Опис інтерфейсу користувача

Взаємодія з інтерфейсом відбувається у контексті проекту. Кожен проект являє собою опис програмних процесів користувачів та доступної фізичної інфраструктури. На рисунку 3.7 зображений приклад сторінки інтерфейсу, що використовується для налаштування проекту.

На рисунку 3.8 зображено сторінку конфігурації програмного сервісу. Необхідно вказати назву процесу, рівень ізоляції кожного процесу, кількість одночасно запущених реплік та вимоги до обчислювальних ресурсів.

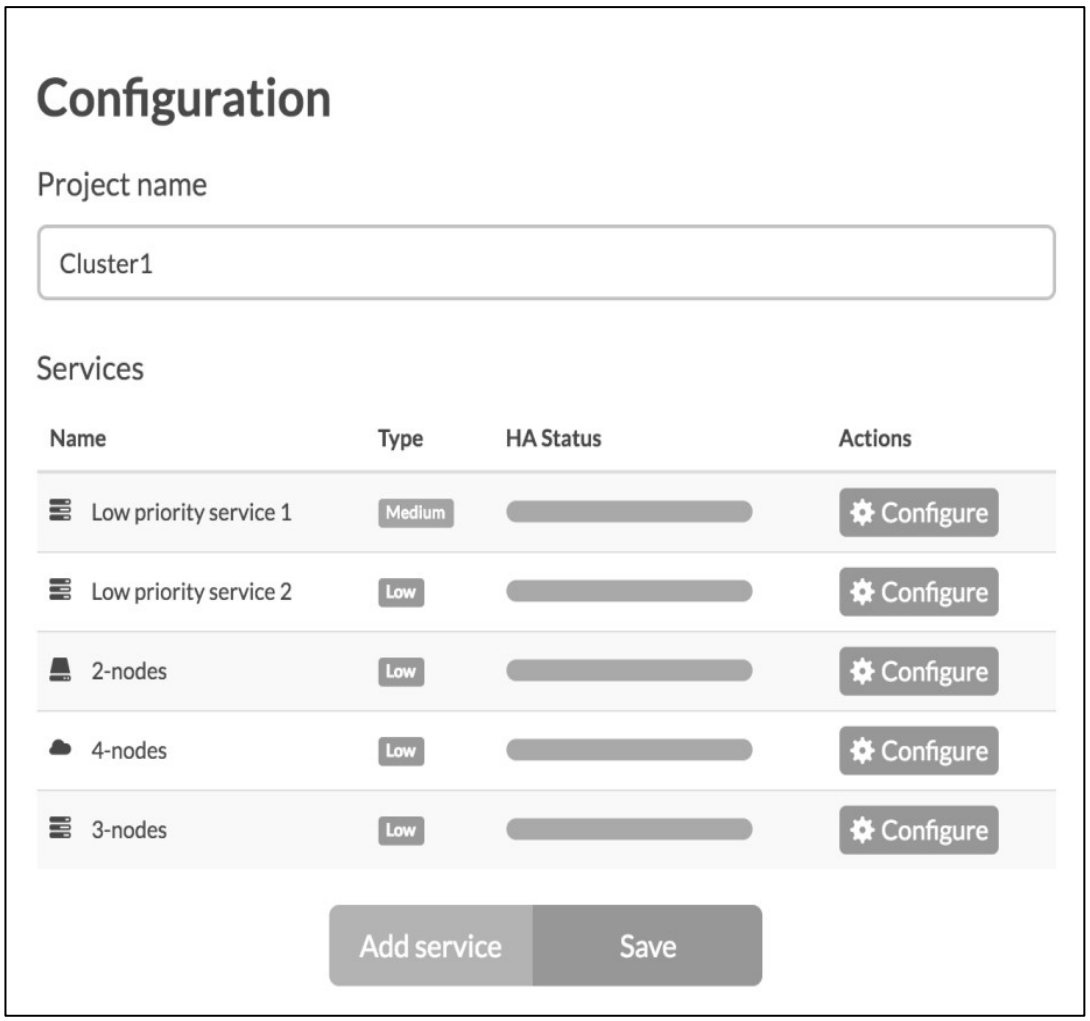

Рис. 3.7. Сторінка налаштування проекту

Рівень ізоляції вибирається в залежності від вимог користувача та визначає розміщення реплік відносно одна одної:

- рівень «critical» – репліки будуть розміщені у різних дата-центрах; - рівень «high» – репліки будуть розміщені на різних фізичних машинах; - рівень «medium» – репліки будуть розміщені на різних віртуальних машинах; - рівень «low» – відсутність вимог до ізоляції.

Панель управління проекту надає змогу користувачу виконувати адміністративні дії щодо користувацьких сервісів, наприклад, ініціювати їх розміщення. Також існує можливість відслідкувати стан сервісів на поточних момент: кількість розміщених сервісів та реплік (процесів).

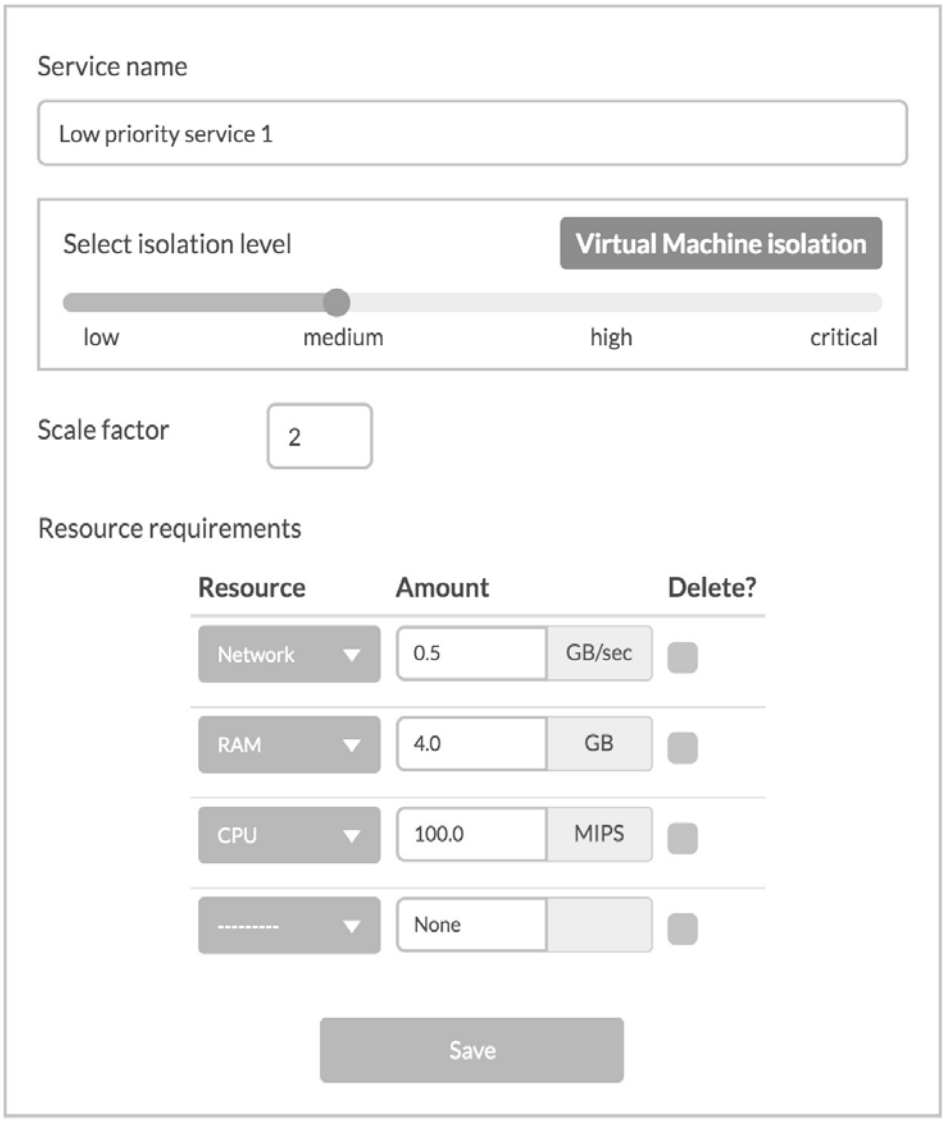

Рис. 3.8. Сторінка налаштування сервісу

Існує 2 режими розіщення: простий (використовуючи налаштування за замовчанням) та складний (із заданням параметрів алгоритму розміщення на різних етапах). Приклад зображений на рисунку 3.9.

Також користувачу надається можливість здійснювати моніторинг стану інфраструктури. На рисунку 3.10 для кожної фізичної машини відображаються список розміщених на ній реплік сервісів, кількість розміщених віртуальних машин, наявний обсяг обчислювальних ресурсів та їх загальна утилізація.

Рис. 3.9. Панель управління проекту

У складному режимі виконаня розміщення існує можливість вибрати тип та параметри алгоритму на двох етапах: під час розміщення серед фізичних машин інфраструктури, та під час розміщення серед віртуальних машин. На рисунках 3.10 та 3.11 зображені приклади налаштування та результати роботи у виглядів графіків фітнес-функції.

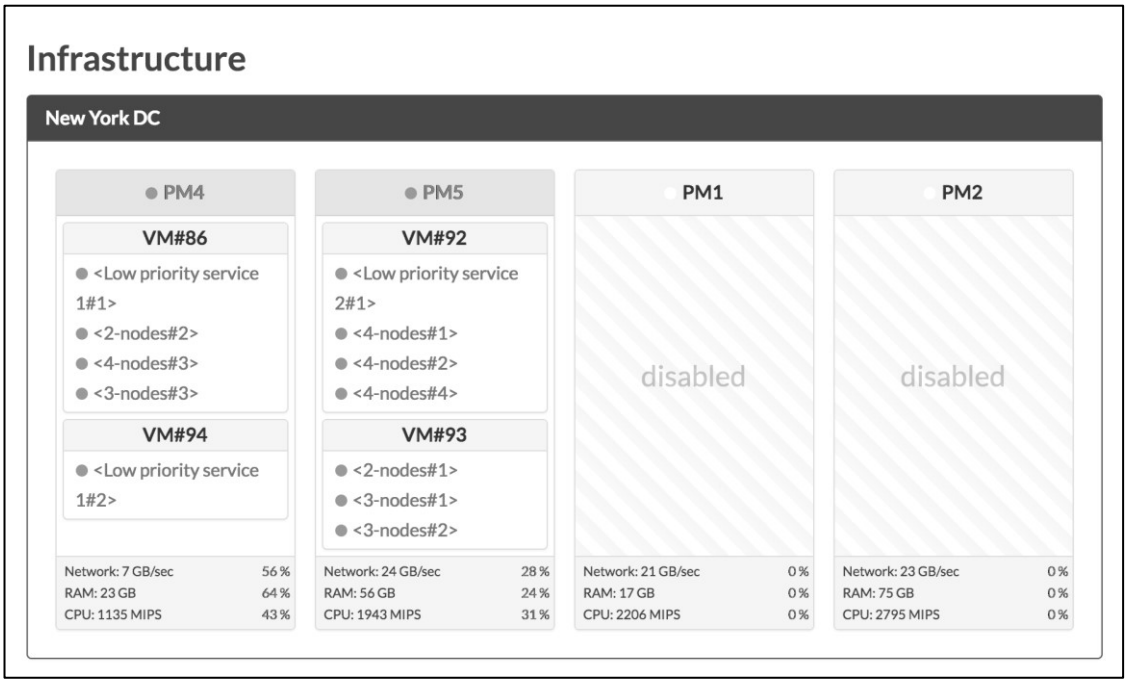

Рис. 3.10. Моніторинг стану інфраструктури

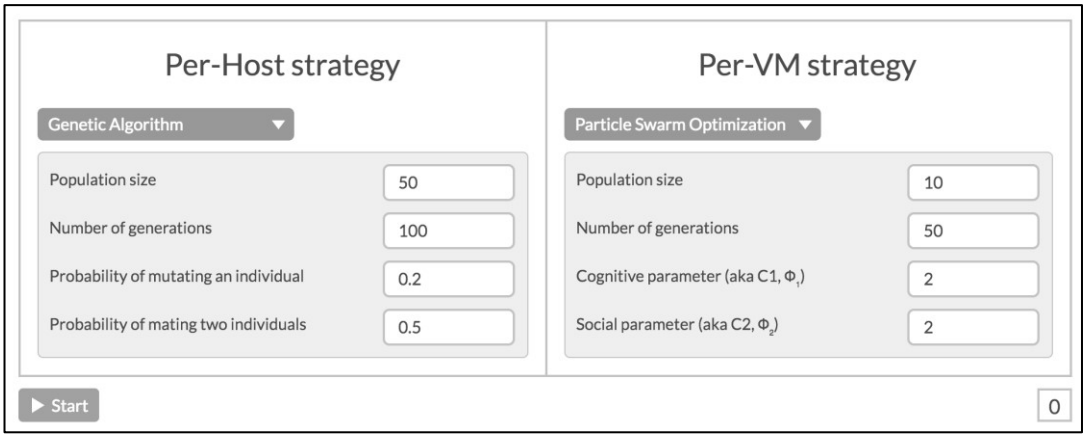

Рис. 3.11. Налаштування алгоритмів розміщення

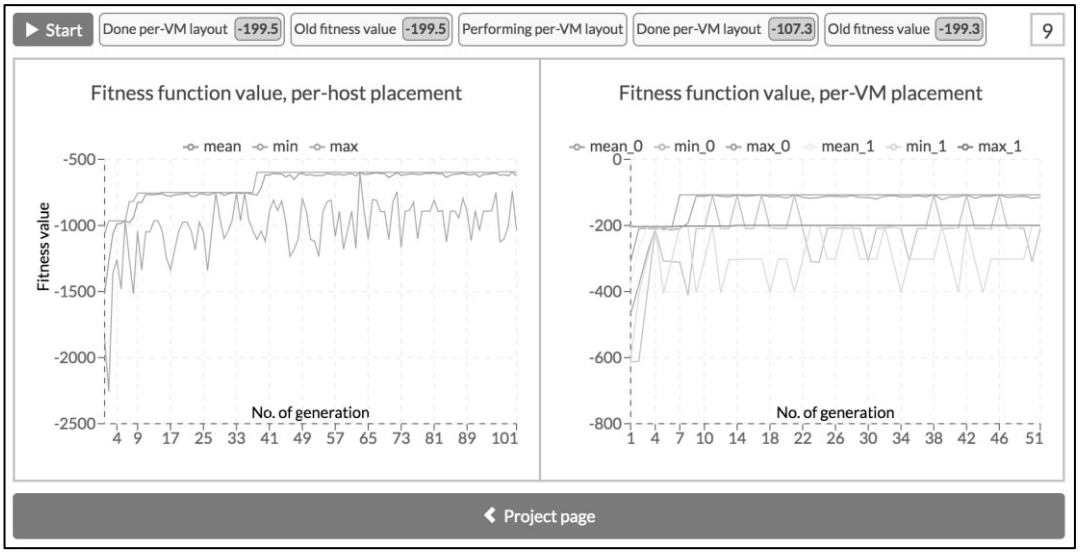

Рис. 3.12. Візуалізація роботи алгоритмів розміщення

3.3. Тестування програмного забезпечення

Для порівняння алгоритмів, ми визначили, як змінюється алгоритм. Результати експерименту наведені на рисунку 3.14.

Таблиця 3.2.

Розподіл ресурсів сервісів, сценарій 2

Ресурс	Межі ресурсу
CPU	250 Mips – 1500 Mips
RAM	0.5 GB – 5 GB
Network	0.5 Gbps – 1.5 Gbps

Таблиця 3.3

Розподіл ресурсів ФМ, сценарій 2

Ресурс	Межі ресурсу
CPU	1000 Mips – 3000 Mips
RAM	2 GB – 16 GB
Network	1 Gbps – 5 Gbps

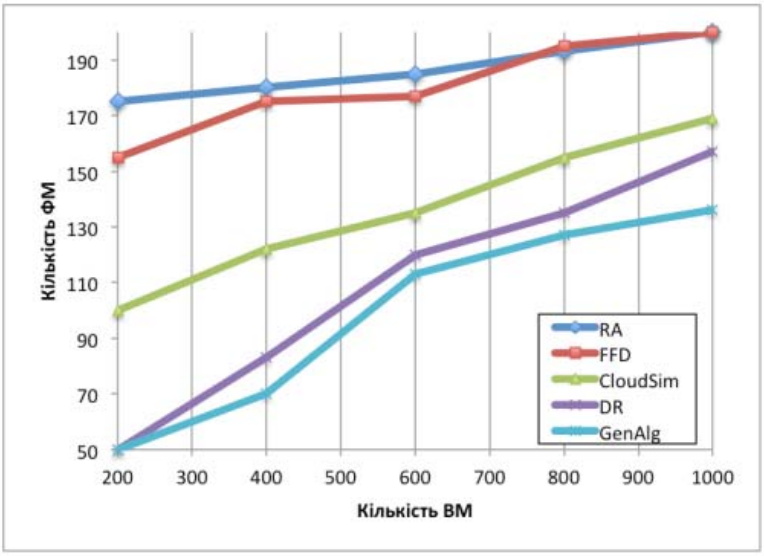

Рис. 3.14. Кількість використаних ФМ у сценарії 2: алгоритми “Random Allocation”, “First Fit Decreasing”, “Demand Risk”, “CloudSim” та “Genetic Algorithm”.

У обох тестових сценаріях алгоритми розміщення RA та FFD демонструють найгіршу ефективність розміщення. Вбудований алгоритм CloudSim знаходить дещо кращі розміщення, в той час як алгоритми DR та GenAlg знаходять рішення близькі до оптимальних.

В усіх тестових сценаріях відбувалась оптимізація розміщення за трьома ресурсами: обчислювальна здатність у кількості інструкцій на секунду (CPU ips), об’єм оперативної пам’яті (RAM) та пропускна здатність мережі (Network).

Об’єми використаних ресурсів фізичних машин для кожного значення кількості сервісів наведені на рисунках 3.15 та 3.16.

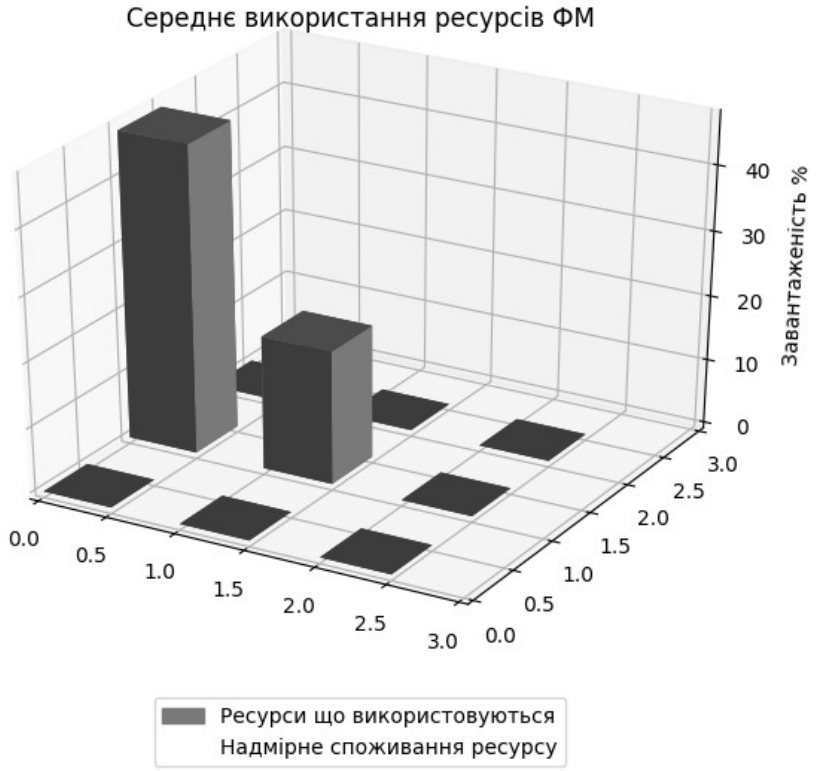

Рис. 3.15. Середнє використання ресурсів для значення X_1

Результати моделювання звідчать про ефективність алгоритму: для розміщення малої кількості сервісів алгоритм намагається використати якомога меншу кількість фізичних машин. Сервери, у яких відсутнє навантаження, можна вимкнути і таким чином зменшити споживання електроенергії у ЦОД. Зі збільшенням кількості сервісів алгоритм використовує для розміщення все більшу кількість серверів.

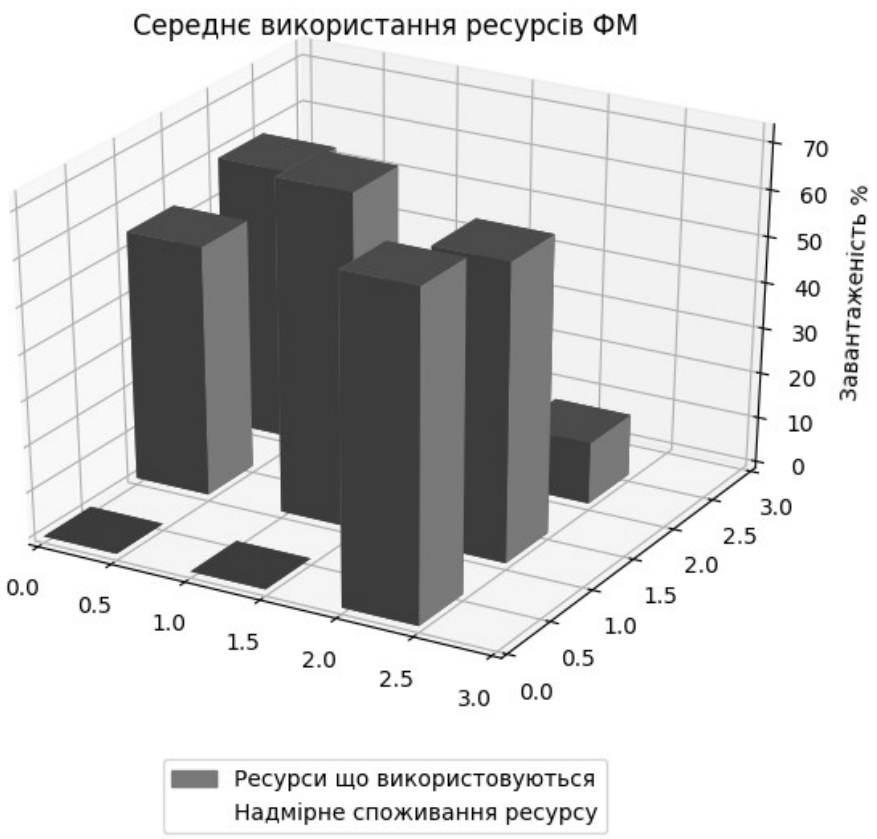

Рис. 3.16. Середнє використання ресурсів для значення X_{20}

Для перевірки виконання умови (2.6) ми використовували генетичний алгоритм з параметрами, наданими у попередньому розділі. Організація сервісів є наступною:

- 3 кластери розміру 1 сервіси (№1, №2, №3);
- 1 кластер розміру 2 (сервіси №4, №5);
- 1 кластер розміру 4 (сервіси №6, №7, №8, №9);
- 1 кластер розміру 3 (сервіси №10, №11, №12).

В обох тестових сценаріях відбувалась оптимізація розміщення за трьома ресурсами: обчислювальна здатність у кількості інструкцій на секунду (CPU ips), об’єм оперативної пам’яті (RAM) та пропускна здатність мережі (Network). Тестові сценарії відрізняються лише кількістю наявних фізичних серверів.

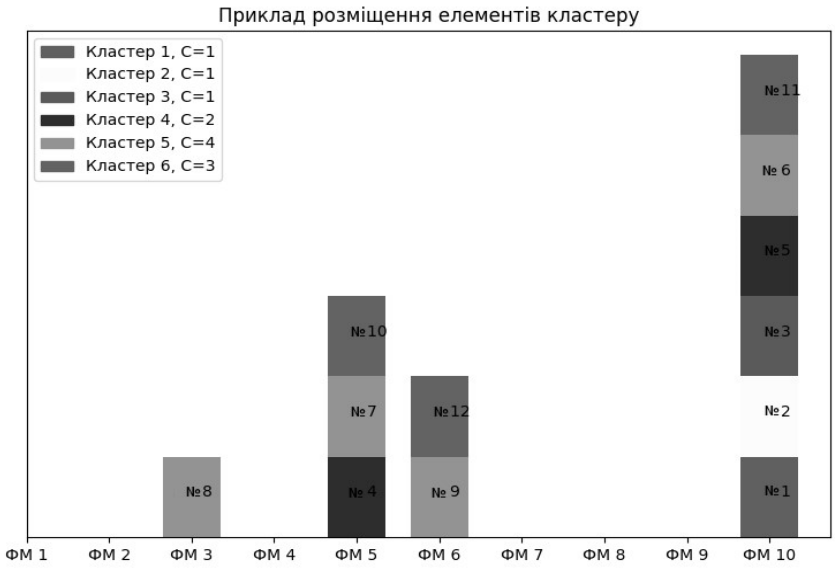

Рис. 3.17. Приклад розміщення кластеру для 10 ФМ

Результати моделювання звідчать про ефективність алгоритму: у обох тестових сценаріях після виконання розміщення сервіси, що входять до складу одного кластеру знаходяться на різних ФМ. У той же час, зі збільшенням кількості наявних серверів, алгоритм використовує лише необхідну йому частину.

Висновки до третього розділу

Програмне забезпечення надає можливість здійснювати моніторинг стану та ініціювати перерозподіл ресурсів елементів інфраструктури, надаючи простий інтерфейс користувача що вимагає мінімальних знань від оператора.

Було проведено порівняння розроблених алгоритмів з існуючими та підтвердженно високу якість отриманих результатів. Результати моделювання свідчать про вдале застосування генетичного алгоритму.

ВИСНОВКИ

В результаті виконання магістерської роботи отримано наступні теоретичні та практичні результати.

В роботі проведено аналіз існуючих проблем в управлінні ресурсами програмних проектів, встановлено можливі способи їх вирішення. Зазначено важливість проблеми створення інструментарію для технічної підтримки IT-інфраструктури. Було розглянуто декілька існуючих технологій управління IT-інфраструктурою, та враховуючи їх переваги та недоліки, було обгрунтовано доцільність створення власної технології управління ресурсами.

Розглянуто значну кількість існуючих моделей хмарних обчислень у контексті управління IT інфраструктурою. Розроблено концепцію інформаційної технології управління IT інфраструктурою на основі ресурсного підходу. Запропонована концепція відрізняється універсальністю застосування, орієнтацією на використання математичних моделей, простотою та ефективністю.

Автором запропоновані способи застосування технологій штучного інтелекту для вирішення задачі розподілу ресурсів. Запропоновано модифікований варіант фітнес-функції, що дозволяє здійснювати оптимальний розподіл елементів інфраструктури, враховуючи їх поточне розміщення. Продемонстровано метод, що легко дозволяє забезпечити виконання вимог високої доступності. Було проведено порівняння розроблених алгоритмів з існуючими та підтвердженно високу якість отриманих результатів.

Було розроблено програмну модель платформи управління IT-інфраструктури, що дозволяє забезпечити вимоги оперативності, надійності та гнучкості. Програмне забезпечення надає можливість здійснювати

моніторинг стану та ініціювати перерозподіл ресурсів елементів інфраструктури, надаючи простий інтерфейс користувача що вимагає мінімальних знань від оператора.

Загалом в магістерській роботі повністю вирішена поставлена задача, розглянуто всі супутні завдання та вирішено проблеми, які виникали в ході виконання завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Глоба Л.С. Математичні основи побудови інформаційно-телекомунікаційних систем. / Л.С. Глоба, М.А. Скуліш, О.М. Дяденко. – К.: Норіта-плюс, 2007. − 348 с.

2. М. Горин. Корпоративный ЦОД: за рамками технологий / М. Горин // Connect! Мир Связи. – 2007. – №8. – С. 19–20.

3. И. Кириллов. Коммерческие ЦОД в Украине: новый этап развития [Електронний ресурс] / И. Кириллов // Сети и бизнес – 2010. – №3 (52). – Режим доступу до журн.: http://www.sib.com.ua/arhiv_2010/2010_3 /statia_3 _1_2010/statia_3_1_2010.htm

4. Cloud Computing Synopsis and Recommendations. Recommendations of the National Institute of Standards and Technology / L. Badger, T. Grance, R. PattCorner, J. Voas. – Special Publication 800-146. – NIST, 2012. – 81 p.

5. Биберштейн Н. Компас в мире сервис-ориентированной архитектуры (SOA) / Биберштейн Н., Боуз С.; пер. с англ. Лунин С. – М.: КУДИЦ-Пресс. – 2007. – 256 с.

6. Krafzik D. Enterprise SOA: Service-Oriented Architecture Best Practices / Krafzik D., Banke K., Slama D. – Prentice Hall Professional, 2005. – 382 p.

7. Joseph J. Grid Computing / Joseph J., Fellenstein C. – Prentice Hall Professional, 2004. – 378 p.

8. Nabrzyski J. Grid Resource Management: State of the Art and Future Trends / J. Nabrzyski, J.M. Schopf, J. Węglarz. – Springer, 2004. – 574 p.

9. Goldworm B. Blade servers and virtualization: transforming enterprise computing while cutting costs / B. Goldworm, A. Skamarock. – Wiley Publishing, Inc., 2007. – 384 p.