TELECOMMUNICATION SYSTEMS AND NETWORKS

DOI: 10.20535/2411-1031.2018.6.2.153497
УДК 004.75

ЯРОСЛАВ ДОРОГИЙ,
ОЛЕНА ДОРОГА-ІВАНЮК,
ДМИТРО ФЕРЕНС

МОДЕЛЬ РОЗПОДІЛУ РЕСУРСІВ КРИТИЧНОЇ ІТ-ІНФРАСТРУКТУРИ З ЧІТКИМИ ПАРАМЕТРАМИ НА ОСНОВІ ГЕНЕТИЧНОГО АЛГОРИТМУ

Проведено детальний аналіз методів та алгоритмів розміщення ресурсів віртуалізованих ІТ-інфраструктур. Розглянуто класичну модель хмарних сервісів, яка складається з трьох шарів. Показано, що специфіка задач, що виконуються в критичних ІТ-інфрастуктурах, ставить перед розробником підвищені вимоги щодо надійності, безпеки, доступності. Встановлено доцільність використання для реалізації створюваної моделі сервісу IaaS. Проаналізовано основних постачальників даного хмарного сервісу, визначено їх переваги та недоліки, обрано найкращого кандидата для реалізації. Наведено детальний опис математичної моделі розподілу ресурсів критичної ІТ-інфраструктури з чіткими параметрами та її використання у поєднанні з генетичним алгоритмом. Описано модель управління віртуальними машинами при серверній віртуалізації з метою подальшого розподілу ресурсів. На прикладі показано, яким чином вона використовується для вирішення поставленої проблеми та як можна оптимізувати та пришвидшити її роботу. Розкрито розписаний генетичний алгоритм, принцип побудови фітнес-функції та його основні операції для розв'язання поставленої задачі. Запропонований генетичний алгоритм у більшій мірі схожий на традиційні генетичні алгоритми. На початку роботи алгоритму випадковим чином створюється початкова популяція рішень-індивідів. Далі, на кожній ітерації алгоритму обчислюється значення функції пристосованості кожного індивіду, для кожного індивіда у популяції вибирається пара для генерації індивідів наступної популяції. Після цього застосовується операція мутації. Окрім того, виконується пошук найкращого індивіду нової популяції та порівнюється із найкращим індивідом попередньої популяції. І на завершення, для побудованої моделі наведено ряд уточнень, які дозволяють використати дану модель для критичної ІТ-інфраструктури з врахуванням вимог високої доступності, таких як відмовостійкість (здатність системи до подальшої роботи після відмови одного із її елементів), неперервна доступність (здатність системи до безперервного обслуговування, незалежно від часу відмови вузлів системи) та високодоступність (здатність системи до подальшої роботи після відмови одного із вузлів, з можливими перервами у роботі). В останній частині статті наведено експериментальні дослідження запропонованої моделі розподілу ресурсів критичної ІТ-інфраструктури з чіткими параметрами на базі генетичного алгоритму.

Ключові слова: архітектура, хмарні послуги, хмарні сервіси, розподіл ресурсів, генетичний алгоритм, критична ІТ-інфраструктура.

Постановка проблеми. Віртуалізація ресурсів стала широко поширеною технологією надання доступу до обчислювальних ресурсів. Головним чином, це обумовлено тим, що віртуалізовані системи можуть забезпечити практично будь-який вид обслуговування і звільнити клієнтів від необхідності створення фізичних інфраструктур.

У зв'язку зі збільшенням попиту на віртуалізовані сервіси, оптимізація використання ресурсів віртуалізованих ІТ-інфраструктур стає ще більш важливою, оскільки її вирішення дасть змогу підвищити продуктивність, доступність та надійність таких систем. Технології віртуалізації виявились досить зручними для таких проблем. Віртуалізація серверів дозволяє

ефективно розподіляти фізичні обчислювальні ресурси між застосуваннями 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.

Різні користувачі можуть мати різні вимоги щодо об'єму обчислень, пам'яті або розміру сховища даних. Фізичні сервери також можуть мати різні можливості. Це приводить до проблеми оптимізації, також відомої як проблеми розміщення віртуальних машин з врахуванням критичності ресурсів, які потрібно на них розмістити. Вирішення цієї проблеми дасть змогу підвищити використання фізичних ресурсів, знизити споживання електроенергії, підвищити надійність IT-інфраструктур. Особливо актуальною дана проблема є для критичних IT-інфраструктур, де вимоги до надійності функціонування є критичними.

Аналіз останніх досліджень. Програмування з обмеженнями – техніка що часто використовується для розміщення віртуальних машин в віртуалізованих середовищах; найбільш вигідна для комбінаторних задач, де рішення повинні задовольняти декільком обмеженням щодо змінних. Зазвичай, задані обмеження можуть бути легко доповнені для задовільнення додаткових потреб. Провайдери мережевих інфраструктур намагаються автоматизувати менеджмент віртуальних машин, враховуючи результуючу якість надання послуг. Така проблема може бути виражена у вигляді задачі оптимізації з обмеженнями. Функція, що оптимізується, вибирається таким чином, щоб досягти повного задовільнення вимог до якості сервісу, за допомогою мінімальних витрат на обслуговування. Значення функції являє собою числене представлення ступені задовільнення вимог щодо якості обслуговування. Подібні алгоритми розміщення віртуальних машин складаються з двох модулів:

– модулю локальних рішень: кожен сервіс віртуалізованої IT-інфраструктури асоціюється з функцією, значення якої дорівнює ступені задовільнення якості надання ресурсів цьому сервісу;

– модулю глобальних рішень: за допомогою локальних рішень усіх сервісів у віртуалізованій IT-інфраструктурі, виконується пошук оптимального значення глобальної функції.

При локальній оптимізації, необхідно врахувати декілька обмежень:

– кількість віртуальних машин кожного класу повинна відповідати вимогам фізичної машини;

– кожен сервіс IT-інфраструктури встановлює верхню межу на кількість віртуальних машин кожного класу та загальну кількість віртуальних машин.

Під час глобальної оптимізації, необхідний об'єм кожного ресурсу виділених віртуальних машин не повинна перевищувати максимальний об'єм ресурсу фізичної машини. Основною ціллю є зменшення кількості активних фізичних машин та повне забезпечення ресурсами всіх критичних процесів.

Оскільки задача є NP-складною [1], застосування повного перебору можливе лише для невеликих розмірностей. Для інших випадків, відома велика кількість евристичних алгоритмів [2] - [8], що демонструють хороші результати:

Best Fit (Найліпший підходящий);

First Fit (Перший підходящий);

Best Fit Decreasing (Найліпший підходящий за спаданням);

First Fit Decreasing (Перший підходящий за спаданням);

Heaviest First (Перший найбільший);

Worst Fit (Найгірший підходящий).

Залежно від мети розміщення, алгоритми розміщення можуть бути поділені на 2 категорії:

– алгоритми оптимізації споживання енергії: пошук розміщення що дозволить системі споживати найменшу кількість енергії;

– алгоритми оптимізації якості обслуговування (QoS - Quality of Service): пошук розміщення що дозволить системі задовільнити вимоги користувачів щодо якості обслуговування.

Залежно від необхідності виконувати міграції (переміщення віртуальної машини від однієї фізичної машини до іншої), алгоритми розміщення поділяються на категорії:

– статичне розміщення: алгоритм розміщення не враховує стани віртуальних та фізичних машин

– динамічне розміщення: алгоритм розміщення виконує пошук оптимального рішення, враховуючи поточне розміщення віртуальних машин.

В свою чергу, динамічні алгоритми розміщення поділяються на 2 типа:

– проактивне: розміщення та міграції віртуальних машин виконуються до моменту, коли система набуде певного стану або до виконання певного критерію

– реактивне: розміщення та міграції віртуальних машин виконуються у відповідь на виконання певного критерію або на досягнення певного небажаного стану. Наприклад, зміна розміщення або виконання міграцій можуть бути необхідні під час технічного обслуговування фізичних машин, або після реєстрування погіршення якості надання послуг.

Система pMapper [3] досягає оптимальних витрат енергії шляхом зменшення кількості міграцій під час пакування віртуальних машин у фіксовану кількість фізичних машин. У дослідженні [6] запропонований одно цільовий алгоритм, що базується на MMAS (Max-min ant system) метаевристиці завдяки чому використовується мінімальна кількість фізичних машин.

У дослідженні [9] були наведені алгоритми розміщення, які максимізують значення метрики під назвою “задоволення” (satisfaction), що відображає відносну придатність фізичної машини до розміщення віртуальних машин на ній. У дослідженні [10] запропонований евристичний підхід для знаходження перевантажених вузлів в датацентрах.

Алгоритм у [11] знаходить оптимальні переміщення віртуальних машин у випадках відмови фізичних машин.

У дослідженні [12] запропоноване рішення під назвою “Backward Speculation Placement” – при виборі переміщень використовуються статистичні дані навантаження та метрика під назвою “ризик попиту”.

Усі розглянуті алгоритми та методи спроектовані для оптимізації фіксованого набору ресурсів, таких як об’єм оперативної пам’яті, використання жорсткого диску або час CPU. Неможливість гнучко налаштовувати набір ресурсів, а також відсутність поділу ресурсів за їх критичністю, робить більшість розглянутих методів непрактичними для використання в критичних IT-інфраструктурах. Окрім того, наведені алгоритми розміщення не допускають введення додаткових специфічних умов оптимізації, що унеможливлює використання їх на практиці.

Метою статті є створення нової моделі розподілу ресурсів для критичних IT-інфраструктур.

Виклад основного матерілу досліджень. Постачальники хмарних сервісів надають широкі можливості користувачам для впровадження хмарних обчислень. Взагалі серед хмарних обчислень можна виділити три основні моделі [4] (див. рис. 1):

IaaS (Infrastructure as a Service);

PaaS (Platform as a Service);

SaaS (Software as Service).

Розглянемо коротко кожну з цих моделей:

– IaaS зазвичай надає уніфіковані апаратні і програмні ресурси, але в деяких випадках і на інфраструктурному рівні для установки ПЗ з оплатою «pay as you go» (по мірі використання). Замовлена інфраструктура може динамічно масштабуватися. На базі такого підходу побудовані Amazon EC2 (Elastic Cloud Computing) Service і Amazon S3 (Simple Storage Service);

– PaaS надає більш високий рівень сервісу, що дозволяє розробляти, тестувати і впроваджувати власні програми. Вбудована масштабованість накладає обмеження на тип розроблюваного програмного забезпечення. Яскравим прикладом реалізації такого підходу є сервіс Google App Engine, що дозволяє впроваджувати Web-застосунки на тій же системі, на якій працюють власні застосунки Google;

– SaaS пропонує готове спеціалізоване ПЗ, що веде до спрощення використання застосунків і до зменшення витрат на розробку. Одним з чудових прикладів реалізації за таким підходом є Salesforce та її онлайнова система управління відносинами з клієнтами. Дослідження компанії Forrester показують, що серед різних варіантів хмарних середовищ переважає SaaS.

Рисунок 1 – Моделі обслуговування користувачів при наданні послуг хмарних обчислень

В табл. 1 наведено порівняння моделей даних хмарних технологій.

Таблиця 1 – Порівняльна характеристика моделей хмарних обчислень з різними правами доступу 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	+	-	-	-
Сервера	+	-	-	-
Сховище	+	-	-	-
Нетворкінг	+	-	-	-

В даній статті буде розглядатися тип хмарних сервісів IaaS (Infrastructure as a Service), оскільки створювана модель повинна мати прямий доступ до апаратних та програмних ресурсів з метою подальшого оптимального їх розподілу.

Хмарні обчислення IaaS. Хмарний сервіс IaaS (інфраструктура як сервіс) являє собою найбільший сегмент ринку хмарних обчислень. В подальшому будуть розглянуті публічні хмарні сервіси IaaS. Публічні хмарні обчислення IaaS в контексті даного дослідження визначаються як стандартизовані, в високій мірі автоматизовані застосування множинної

аренди (з великою кількістю користувачів), де обчислювальні ресурси, будучи доповнені мережевими можливостями та функціями сховища, належать та організовуються надавачем сервісів та пропонуються клієнту за вимогою. Клієнт має можливість самостійного керування цією інфраструктурою з використанням графічного користувацького веб-інтерфейсу, що служить в якості консолі керування IT-операціями в конкретному середовищі оточення. Опціонально може пропонуватися інтерфейс програмних застосувань до інфраструктури (Application Programming Interface (API)).

"Інфраструктура як послуга" (IaaS) – це комплексна IT – інфраструктура, що подається у вигляді послуги. Кожен користувач чи клієнт отримує доступ до частини консолідованого пулу об'єднаних ресурсів для створення та використання власної обчислювальної інфраструктури у відповідності з потребами.

Варто більш детально розглянути переваги моделі IaaS перед іншими моделями. Насамперед це:

– підвищена ефективність – віртуалізовані ресурси об'єднуються в пули, забезпечуючи використання всієї ємності фізичної інфраструктури;

– підвищення оперативності – IT-ресурси можна виділяти за вимогою та швидко повертати назад в пул;

– швидке масштабування – миттєве виділення додаткових ресурсів у відповідності з бізнес-вимогами в періоди пікових навантажень, а також при збільшенні чи зменшенні розміру організації;

– зниження затрат – модель "оплата по мірі використання" дає змогу скоротити витрати на інфраструктуру, електроенергію та обслуговування;

– підвищення продуктивності роботи IT-служби – автоматизоване виділення ресурсів через портал самообслуговування;

– скорочення не використовуваних ресурсів – прозорі методи ціноутворення, вимірювання та розподілу витрат між підрозділами дають змогу IT-адміністраторам виявити потенційні області скорочення затрат;

– підвищення ефективності інвестицій в IT-інфраструктуру;

– підвищення рівня безпеки та захисту інформаційних ресурсів.

Порівняльна характеристика основних провайдерів IaaS. На ринку послуг IaaS найбільшої популярності здобули такі провайдери, як: Amazon Web Service EC2, Google Compute Engine та Microsoft Azure. Для них і виконано порівняльну характеристику.

Провайдер Amazon Web Service. AWS включає в себе обслуговування ядра обчислень Amazon, що надає змогу користувачам налаштовувати віртуальні машини з використанням попередньо сконфігурованих (за замовчуванням) чи користувацьких MAC (шаблонів віртуальних машин). В даному випадку користувач має можливість обрати розмір, потужність, обсяг пам'яті і число віртуальних машин з різних регіонів та призначити для них зони доступності, в межах яких можна запускати дані машини. AWS надає також можливості балансування навантаження (ELB) та автоматичного масштабування. ELB розподіляє навантаження між екземплярами машин для підвищення продуктивності.

APM забезпечує ефемерне (тимчасове) зберігання, при якому створюється лише один екземпляр сеансу і який руйнується при завершенні роботи. AWS також пропонує зберігання об'єктів за допомогою S3 служби та послуги архівування. AWS повністю підтримує реляційні і NoSQL бази даних, великі обсяги даних.

Віртуальні приватні хмари Amazon (VPCs) дозволяють користувачам створювати групи віртуальних машин в ізольованих мережах в хмарі. За допомогою VPCs користувачі можуть визначати топологію мереж, створювати підмережі, таблиці маршрутизацій, приватні діапазони адрес і мережеві шлюзи.

Amazon Web Service має досить зручний інтерфейс для контролю за станом хмари, кількістю розгорнутих віртуальних машин, логічних дисків.

AWS приваблює клієнтів методом оплати за використання ресурсів хмари. Користувач платить тільки за «округлену» кількість годин, впродовж яких він використовував дану систему. Мінімальна одиниця часу використання – одна година.

Провайдер Google Compute Engine. У 2012 році компанія Google представила свій хмарний сервіс обчислень – Google Compute Engine (GCE). Google Compute Engine дозволяє користувачам запускати віртуальні машини так само, як AWS, в різних регіонах та зонах доступності. Однак, GCE не була широко доступна для користувачів до 2013 року. Лише з 2013 року Google почала проводити свої власні удосконалення, такі як балансування навантаження, розширена підтримка операційних систем, жива міграція віртуальних машин. Інтерфейс користувача для Google Compute Engine наведено на рис. 2.

Хмарна платформа Google, так само забезпечує як тимчасове, так і постійне зберігання даних. Для зберігання об'єктів, GCP має Google Cloud Storage. GCP підтримує реляційну БД через Google Cloud SQL. Nearline Google пропонує архівування практично без затримки на відновлення даних.

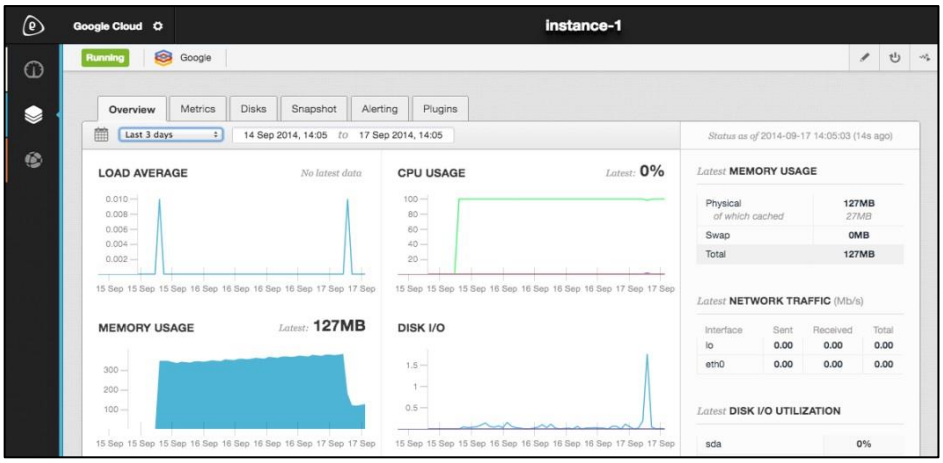

Рисунок 2 – Інтерфейс користувача в Google Compute Engine

Провайдер Microsoft Azure. У 2012 році компанія Microsoft представила свою службу обчислень, для попереднього тестування, але не зробила її загальнодоступною, доки користувачі Azure в травні 2013 року не обрали VHD (Virtual Hard Disk), що дуже схоже на AMI Amazon при створенні віртуальних машин. Віртуальний жорсткий диск може бути визначений за замовчуванням, або самим користувачем. Для кожної віртуальної машини необхідно вказати кількість ядер і об'єм виділеної пам'яті. Інтерфейс користувача в Microsoft Azure наведено на рис. 3.

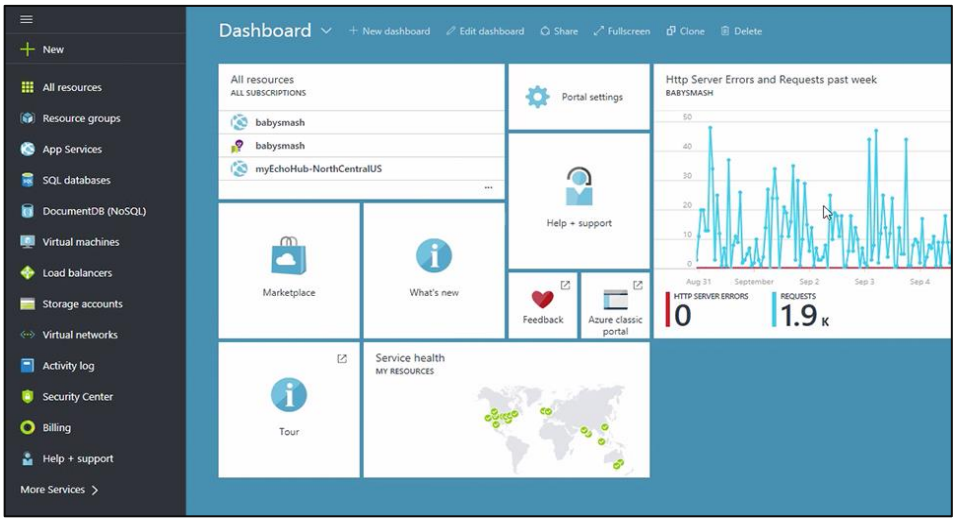

Рисунок 3 – Інтерфейс користувача в Microsoft Azure

Azure використовує тимчасове сховище (D диск) і Page Blobs (опція Block Storage від Microsoft) для томів віртуальних машин. Блок Blobs і файли служать для зберігання об'єктів. Azure підтримує реляційні, NoSQL бази даних і великі дані за допомогою таблиць Windows Azure і HDInsight. Результати порівняння провайдерів наведені в табл. 2.

Таблиця 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, округлюється (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

Модель розподілу ресурсів IT-інфраструктури з чіткими параметрами. Для впровадження моделі розподілу ресурсів необхідно розглянути декілька ознак, що можуть вплинути на тип самої моделі. Моделі можуть відрізнятися в залежності від методів ведення бізнесу (для власних бізнес-процесів, чи для зовнішніх клієнтів). Також на вибір моделей впливає тип архітектури IT-інфраструктури. Залежно від рівня абстракції ресурсів варіюється складність математичної моделі задачі, що розв'язується. Задачі великої розмірності розв'язуються у два етапи: на першому етапі здійснюється розподіл абстрактних ресурсів кожного типу без прив'язки до їх конкретного місцезнаходження з уточненням отриманих результатів на другому етапі.

Дуже істотно впливає на вибір моделей ознака забезпеченості ресурсами, тобто, чи дозволяється часткова підтримка сервісів, чи вони мають підтримуватися у повному обсязі, або не підтримуватися взагалі. У найпростішому випадку дворівневої клієнт-серверної архітектури модель розподілу обмежених ресурсів може подаватись у вигляді декількох бізнес-процесів, що безпосередньо використовують частину одного чи декількох незалежних один від одного ресурсів (див. рис. 4).

Введемо необхідні для побудови моделей наступні позначення:

$Z_1 ... Z_n$ – бізнес-процеси, підтримку яких забезпечує функціонування ІТС;

$W = (w_1, .., w_n)$ – коефіцієнти важливості бізнес процесів $Z_1 ... Z_n$ відповідно;

$R_1 ... R_m$ – інтегровані ресурси ІТС, необхідні для підтримки функціонування бізнес-процесів;

$P = \|p_{ij}\|$ – матриця потреб бізнес-процесів у ресурсах ІТС, де p_{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:

$$\begin{cases} d_{ij} = 1, \textit{ якщо } p_{ij} > 0, \\ d_{ij} = 0, \textit{ якщо } p_{ij} = 0; \end{cases} \quad (1)$$

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

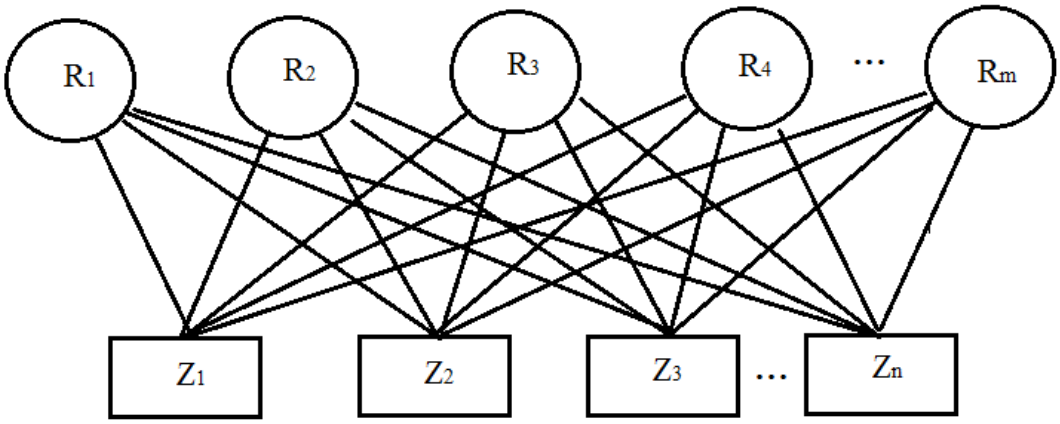

Рисунок 4 – Модель розподілу обмежених ресурсів ІТ-інфраструктури

Розглянемо дискретний випадок, коли бізнес-процес або підтримується в повному обсязі, або повністю блокується. Введемо вектор $X = (x_1, ..., x_n)$, де:

$$x_i = \begin{cases} 1, \textit{ якщо процес } Z_i \textit{ обслуговується}; \\ 0, \textit{ в протилежному випадку.} \end{cases} \quad (2)$$

Тоді критерій оптимального розподілу ресурсів інформаційно-телекомунікаційної системи (ІТС) можна подати у вигляді:

$$\max \sum_{i=1}^{n} x_i w_i . \quad (3)$$

Мають також бути використані обмеження по ресурсам **Помилка! Джерело посилання не знайдено.**4):

$$\sum_{i=1}^{n} x_i p_{ij} \le r_j, \ j = 1, ..., m. \quad (4)$$

Варто зазначити, що наведена вище модель, а також інші моделі, які розглядаються, належать до лінійних або нелінійних, неперервних, булевих та змішаних задач математичного програмування, які до того ж мають стохастичні аналоги. Для детермінованих задач можна використовувати евристичні алгоритми, методи м'яких обчислень та точні методи, використання яких обмежено з огляду на розміри задач. Так, для задач лінійного програмування можна використовувати відомі методи. Задачі булевого програмування можна розв'язувати методами часткового перебору. Точні методи дають змогу знайти найкращий розв'язок, але їх використання можливе лише для задач обмеженої розмірності, оскільки час пошуку рішень суттєво зростає зі збільшенням складності задачі. На відміну від них, евристичні методи та методи штучного інтелекту, насамперед генетичні алгоритми (ГА), дають змогу знайти задовільний розв'язок за доволі короткий час, навіть для

задач великої розмірності, до того ж їхню ефективність можна поліпшити за рахунок врахування особливості задач.

Загальна схема алгоритму подана на рис. 5.

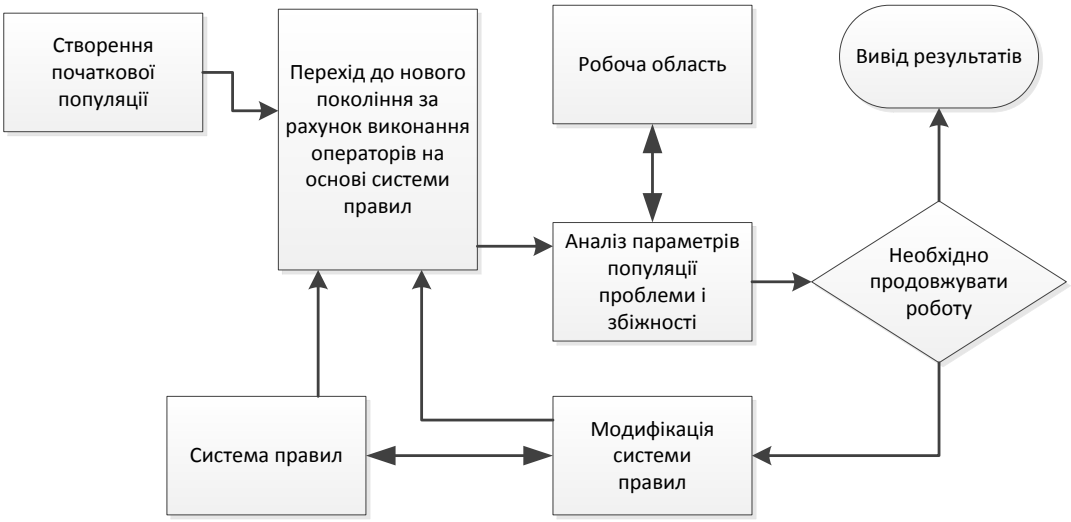

Рисунок 4 – Схема роботи генетичного алгоритму

Використання агентів дає змогу здійснювати управління ресурсами і навантаженням на двох рівнях. Перший рівень – управління окремими віртуальними серверами. Другий рівень – управління окремими застосуваннями, що встановлені на віртуальних серверах. Модуль прогнозування призначений для аналізу трендів з метою подальшої реалізації проактивного управління, у цьому випадку – перерозподілу ресурсів між застосуваннями до того, як виникла їх нестача для застосувань, задіяних у підтриманні бізнес-процесів високої важливості. Модуль планування призначений для довгострокового управління ресурсами IT-інфраструктури у разі її розвитку чи істотних змін у її функціонуванні. Диспетчер запитів слугує для обмеження клієнтського трафіку заданого типу з метою вивільнення ресурсів, споживання яких залежить від навантаження.

Модель управління віртуальними машинами при серверній віртуалізації. Нехай є декілька фізичних серверів $S_i, i = 1,..., n;$ на яких під управлінням гіпервізорів функціонують віртуальні машини (ВМ) $V_j, \ j = 1,..., m$ (див. рис. 6).

Кожна з ВМ в залежності від потоку клієнтських запитів використовує певну кількість ресурсів типу $R_k, k = 1,..., l$ (пам’ять, процесорний час, дисковий простір, пропускна спроможність підсистеми вводу-виводу, зовнішніх каналів зв’язку).

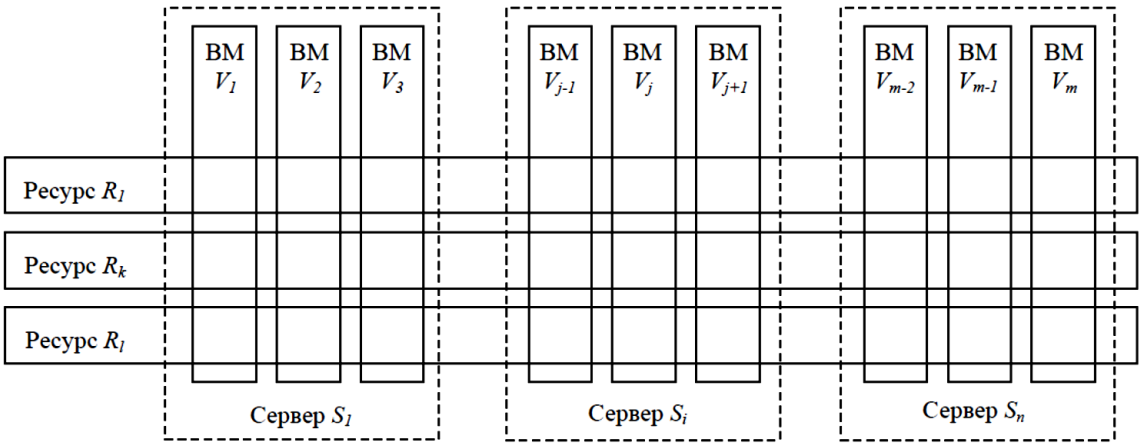

Рисунок 5 – Модель розміщення віртуальних машин на фізичних серверах [13]

Розглянемо процес розподілу ресурсів серверів між віртуальними машинами. Для цього введемо необхідні для побудови моделей позначення:

r_{kj} – кількість ресурсу типу R_k, що встановлено на сервері S_i;

p_{kj} – потреби ВМ V_j у ресурсах типу R_k для забезпечення заданого рівня SLA;

x_{ij} – булева змінна, яка визначає, чи встановлена ВМ V_j на сервері S_i.

Оскільки кожна ВМ одночасно розташована не більше ніж на одному сервері, має виконуватись умова описана у формулі (5):

$$\sum_{i=1}^{n} x_{ij} \le 1,\ j = 1,..., m. \tag{5}$$

До того ж, для нормального функціонування ВМ повинні бути забезпечені достатнім обсягом ресурсів серверів, на яких вони розташовані (6).

$$\sum_{j=1}^{m} x_{ij} p_{kj} \le r_{ki}, k = 1,..., l;\ i = 1,..., n. \tag{6}$$

Розглянемо задачу розташування ВМ для випадків нестачі та надлишку ресурсів.

У разі, якщо внаслідок збільшення клієнтських запитів потреба окремих ВМ у ресурсах певного типу збільшилась настільки, що стає неможливим забезпечити усі ВМ необхідною кількістю ресурсів, природним виходом стає задача підтримки найбільш важливих бізнес-процесів шляхом забезпечення ресурсами тих ВМ, на яких працюють пов'язані з ними сервіси, за рахунок менш важливих.

Позначимо через $w_j, j = 1,..., m;$ важливість застосувань, встановлених на ВМ V_j. Тоді задачу можна сформулювати наступним чином (7)

$$\sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} w_j \to \max \tag{7}$$

при обмеженнях (5), (6).

Ця задача становить собою лінійну задачу булевого програмування. Для її вирішення скористаємося методом гілок та меж. Але спочатку покажемо, як можна, враховуючи специфіку задачі, зменшити початкову кількість можливих комбінацій, яка складає $2mn$.

По-перше, згідно з обмеженням (5), кожна ВМ може бути розташована не більше, ніж на одному сервері, тобто замість перебору $2n$ комбінацій для кожної ВМ V_j, можна обмежитись розглядом лише $n+1$ варіантів: $y_j = 0,..., n;$ де y_j – номер сервера, на якому встановлена ВМ $V_j (y_j = 0)$ у випадку, якщо внаслідок нестачі ресурсів ВМ V_j не встановлена на жодному сервері або ресурси для неї виділяються по залишковому принципу).

По-друге, враховуючи те, що вимоги ВМ у ресурсах p_{ij} є невід'ємними, якщо на якомусь з етапів побудови дерева варіантів розміщення ВМ по серверам одне з обмежень (6) перестає виконуватись, продовжувати побудову цієї гілки немає сенсу, оскільки для усіх її вузлів це обмеження також не буде виконуватись.

По-третє, слід врахувати те, що дуже часто при побудові серверних ферм використовують сервери з однаковою конфігурацією. Якщо, наприклад, є два ідентичних сервера, кількість можливих варіантів розміщення ВМ можна скоротити в 2 рази, оскільки перенесення усіх ВМ з першого сервера на другий, а з другого на перший ніяк не вплине на об'єм ресурсів, які використовуються. Тобто перестановка рядків у матриці $X = \|x_{ij}\|$, які відповідають однаковим стовпцям у матриці $X = \|x_{ij}\|$, ніяк не впливає ані на виконання обмежень (5), (6), ані на значення критерію (7). Іншими словами, для серверів з однаковою конфігурацією важливе не абсолютне розміщення ВМ на цих серверах, а їх розташування одна відносно іншої. При збільшенні кількості серверів з однаковою конфігурацією кількість

“зайвих” комбінацій значно зростає від $(n-1)$ для випадку розміщення усіх ВМ на одному сервері до $(n!-1)$ при розміщенні ВМ на різних серверах.

Так, наприклад, для випадку трьох ВМ і трьох ідентичних серверів з можливих 27 комбінацій розташування ВМ лише 5 є унікальними, а решта – 22 комбінації – «дзеркальними» до них. Тобто, навіть для таких незначних значень m і n початкову множину варіантів можна скоротити більше, ніж у 5 разів.

Для наочності буквами (“A”, “B”, “C”) позначені ВМ, цифрами (1, 2, 3) – номери серверів, комбінацією букви з цифрою – розміщення ВМ на сервері. Алгоритм перебору при побудові дерева розташування ВМ полягає у наступному: на черговому рівні чергової гілки кількість серверів, що розглядаються, для перебору приймається на 1 більше, ніж максимальна для попередніх рівнів цієї ж гілки, але не більше ніж n.

Так, на рівні “A” розглядається лише випадок розташування ВМ “A” на сервері №1, а на серверах №2 та №3 не розглядається, оскільки сервери №1-3 є повністю ідентичними і розташування ВМ “A” на них призведе до тих самих результатів. При розв’язанні задачі в реальних умовах, з метою зменшення кількості міграцій ВМ, при виборі довільного сервера обирається той, на якому ця ВМ вже встановлена. На рівні “B” розглядаються два варіанти “B1” – розміщення ВМ “B” на тому ж сервері, що і ВМ “A”, та “B2” – розміщення ВМ “B” на іншому сервері, у цьому випадку №2. Варіант “B3” не розглядається, оскільки він нічим принципово не відрізняється від варіанту “B2” і призведе до тих самих результатів (аналогічно, якщо ВМ “B” уже встановлена на сервері №3, слід використовувати саме його, а не сервер №2).

Якщо у наявності є сервери з іншою конфігурацією, до схеми перебору додаються відповідні вузли на кожному рівні. Сюди також відносяться випадки, коли внаслідок нестачі ресурсів дозволяється низькоприоритетні ВМ не розміщувати на жодному сервері чи виділяти ресурси для них за залишковим принципом. Формально це описується шляхом розміщення таких ВМ на неіснуючому сервері № 0.

Крім визначення процедури розгалуження, метод гілок і меж передбачає визначення процедур оцінки верхньої та нижньої меж за допомогою наближених, але швидких алгоритмів.

При вирішенні задача максимізації для визначення оцінки зверху можна використати оптимістичний прогноз, який полягає в припущенні того, що на черговому кроці алгоритму усі ВМ, не розподілені між серверами, можуть бути розподілені таким чином, що жодна з них не отримує відмови в обслуговуванні (не буде розміщена на неіснуючому сервері № 0) за умови виконання ресурсних обмежень (6).

Щоб одержати оцінку знизу, пропонується використати жадібний алгоритм, упорядкувавши ВМ за зменшенням вимог до того ресурсу, нестача якого стала причиною переходу від старого до нового плану розміщення ВМ.

Процедура відсікання гілок полягає у тому, що гілка, для якої оцінка верхньої межі менша, ніж оцінка нижньої межі хоча б однієї з інших гілок, розглядається як безперспективна та виключається з дерева рішень. Враховуючи те, що вимоги ВМ до ресурсів є невід’ємними, ця процедура може бути доповнена відсіканням тих гілок, для яких припиняють виконуватись обмеження (6).

У разі, якщо на черговому кроці буде знайдене рішення, при якому усі ВМ успішно розміщені, тобто жодна ВМ не розмішена на сервері № 0, пошук можна припинити, оскільки глобальний оптимум знайдений і критерій (7) набуває максимального значення. Але, за умови наявності часових можливостей, продовження пошуку може допомогти знайти рішення, що забезпечить меншу кількість міграцій ВМ при реалізації нового плану розташування, ніж уже знайдене рішення та його “дзеркальні” рішення.

Застосування технологій штучного інтелекту для вирішення задачі багатомірного пакування. Задачі управління ресурсами можуть бути сформульовані як задачі лінійного програмування та бути вирішені оптимально, знаходження точного рішення зазвичай є

марною тратою ресурсів. Оскільки такі задачі належать до класу NP-повних, простір пошуку рішень є надзвичайно великим, містить значну кількість потенційних рішень та потребує значної кількості обчислювальних ресурсів. На практиці достатньо знайти рішення, що є близьким до оптимального, за короткий проміжок часу.

Сьогодні методи інтелектуального аналізу даних широко застосовуються для розв'язання проблем управління ресурсами. Особливо перспективними в цій галузі видаються генетичні алгоритми [14], які належать до еволюційних методів пошуку екстремуму в проблемах функціональної оптимізації, умовної чи безумовної.

Якщо рішення проблеми відповідним чином закодовані, то заснований на принципах моделювання біологічних процесів генетичний алгоритм здатен покроково поліпшувати початковий набір рішень до одержання прийнятного для дослідника результату.

Проблеми розподілу ресурсів дуже зручно кодувати, оскільки вони найчастіше належать до проблем умовної оптимізації, сформульованих у вигляді задач математичного програмування з булевими змінними. У випадках неперервних або цілочислових змінних або змішаних задач математичного програмування також існують ефективні схеми кодування рішень.

Оскільки проблеми розподілу ресурсів становлять досить окремий клас проблем великої розмірності, то для генетичних алгоритмів для цих проблем важливою умовою є ефективне управління збіжністю. Тобто необхідно мати такий механізм у генетичному алгоритмі, який, виходячи із вимірності задачі і заданої точності рішень, міг знайти потрібний компроміс між кількістю кроків і точністю.

Основи генетичних алгоритмів були закладені Дж. Холандом [14], [15], який змоделював природній відбір за принципом "виживає самий пристосований" протягом декількох поколінь гіпотетичної біологічної популяції.

Розглянемо основні положення генетичних алгоритмів для розв'язання оптимізаційної проблеми, базуючись на бітових представленнях особин популяції, кожна з яких становить можливе розв'язання задачі. Кожна особина оцінюється мірою "пристосовуваності до умов існування", що в термінах оптимізації функцій відповідає перевазі відповідного рішення над іншими за умови виконання обмежень.

У генетичних алгоритмах найчастіше за структуру даних зображення особини вибирають одну або декілька хромосому – бітових рядків. Далі будемо розглядати лише випадок, коли особині відповідає одна хромосома. Кожна хромосома – це комбінація генів, які розташовуються в різних позиціях (локусах) хромосоми і набувають значень, що отримали назву "алелі". Якщо мова йде про бінарні рядки, то гену відповідає біт, локусу – його позиція в рядку, а алелі – значення біта (0 або 1).

Усі можливі екземпляри структури даних проблеми (тобто хромосоми) складають простір пошуку можливих рішень генетичного алгоритму. Іноді ще використовують терміни "генотип" і "фенотип; Перший з них стосується повної генетичної моделі особини і відповідає структурі даних (у нас – хромосоми), а другий – зовнішнім спостережуваним ознакам і відповідає вектору у просторі параметрів проблеми. Кожній точці останнього відповідає точка простору, в якому здійснює пошук генетичний алгоритм, який працює з хромосомами.

Випадковим чином генерується початкова популяція. Після цього генетичний алгоритм виконує однотипні ітерації доти, поки не буде виконана умова зупинки. Кожна ітерація полягає у послідовному виконанні таких операцій: відбір особин попередньої популяції; кросинговер; мутація. Відбір особин найчастіше відбувається на основі ймовірностей: $P(i)$, які обчислюються за формулою (8):

$$P(i) = \frac{f(i)}{\sum_{i=1}^{m} f(i)}, \tag{8}$$

де $f(i)$ – пристосованість особини i.

Потім здійснюється пропорційний відбір n особин згідно з величиною $P_{3(i)}$, наприклад за схемою рулетки Гольдберга [16].

Схрещення найчастіше здійснюється за схемою випадкового розбиття відібраних особин на пари з наступним виконанням з заданою ймовірністю P_c одноточкового схрещення. Для виконання останнього, випадковим чином вибирається точка розриву і утворюється два генотипи нащадків.

Якщо схрещення виконується, то нащадки не замінюють у цій популяції батьків, а замінюють найменш пристосовані особини (це здійснюється шляхом відбору на новій ітерації).

Мутації здійснюються також за імовірнісною схемою, причому Дж. Холанд вважає, що доцільно змінювати значення приблизно 1 на кожних 10000 бітів, включаючи батьків і нащадків. Загалом, мутація виконується для деяких хромосом, що піддаються їй, шляхом зміни кожного біта на протилежний з ймовірністю P_m.

Проблема розміщення віртуальних машин є багатокритеріальною задачею оптимізації. Наша мета полягає у тому, щоб знайти оптимальне розміщення, що являє собою відношення між віртуальними та фізичними машинами. Для формального опису моделі, введемо наступні поняття:

R – множина ресурсів;

P – множина фізичних машин;

V – множина віртуальних машин;

v_i – віртуальна машина i;

v_i^r – необхідність віртуальної машини i у ресурсі r;

P_{all} – множина задіяних (аллокованих) фізичних машин;

p_j – фізична машина j;

p_j^r – наявність ресурсу r у фізичної машини j;

V_i – множина віртуальних машин що призначені p_j.

Метою є мінімізація кількості задіяних фізичних машин, тим саме мінімізуючи використання спожитої енергії:\

$$P_{all} = \{p_j \in P \,|\, |V_j| > 0\}. \tag{9}$$

Опишемо додаткові критерії оптимізації:

Усі віртуальні машини повинні бути розміщені:

$$V = \bigcup_{p_j \in P} V_j. \tag{10}$$

Кожна віртуальна машина може бути розміщена лише на одній фізичній машині:

$$V_i \cap V_j, \; i \neq j. \tag{11}$$

Для кожної фізичної машини, сумарне споживання кожного ресурсу її аллокованих віртуальних машин не повинне перевищувати об’єм доступного ресурсу фізичної машини:

$$p_j^r \geq \sum_{v_i \in V_j} v_i^r, \; r \in R. \tag{12}$$

Кожне рішення у популяції характеризується хромосомою, що має структуру N-мірного вектору:

$$C = (p_1, \; p_2, \; ..., \; p_N), \tag{13}$$

де N – кількість віртуальних машин;

p_i – фізична машина, що призначена i-тій віртуальній машині.

У запропонованій структурі кожна хромосома містить N генів, кожен з яких визначає позицію кожної віртуальної машини. Така структура хромосоми дає можливість простим чином дотримуватись виконання обмежень (10) та (11).

Для досягнення оптимального розподілу усіх ресурсів, визначимо функцію пристосованості кожного індивіда популяції (14), (15) та (16):

$$f(i) = -(k+1)\sum_{r\in R}\sum_{p_j P_{all}} \frac{w_j^r p_j^r}{p_j^r}, \tag{14}$$

$$u_j^r = \frac{\sum_{v_i \in V_j} v_i^r}{p_j^r}, \tag{15}$$

$$w_j^r = 1 - u_j^r, \tag{16}$$

де u_j^r – використання ресурсу r фізичною машиною j;

w_j^r - марнування ресурсу r фізичною машиною j;

k – кількість фізичних машин, для яких споживання ресурсів є надмірним.

Функція пристосованості, що має від'ємне значення, визначає наскільки ефективним є дане розміщення. Неефективне розміщення матиме дуже мале значення функції пристосованості, що дасть змогу вилучати такі розміщення із вибірки. Також, якщо сумарне споживання ресурсу однієї фізичної машини перевищує її запас, ми зменшуємо значення функції пристосованості індивіда у зовнішній сумі (14). Завдяки такому штрафу досягається компроміс між недостатнім та надмірним використанням ресурсів.

У той же час марнування ресурсів є більш сприятливим, ніж їх перевикористання. Таким чином, значення штрафу буде меншим завдяки внутрішній сумі. Таким чином, ми досягаємо мінімального марнування ресурсів в усій інфраструктурі.

Варто зауважити, що штрафи не додаються для фізичних машин, що знаходяться у стані простою (для яких не призначено жодної віртуальної машини), оскільки такі фізичні машини можуть бути вимкнені. Нашою метою є максимальна утилізація ресурсів датацентру, і для реалізації цього ми намагаємось збільшити кількість фізичних машин, які можна вимкнути. Чим більша кількість таких фізичних машин, тим більше значення функції пристосованості , що приводить до зменшеного загального використання енергії.

Під час операції вибірки відбувається попарний відбір рішень, над якими згодом виконується операція схрещення. Операція вибірки створює підпопуляцію з рішеннями із загальної популяції, вибрані випадковим чином. Розмір підпопуляції дорівнює 5% розміру загальної популяції. Це значення було підібрано експериментальною та контролює межу між різноманітністю популяції та якістю отриманих рішень.

Операція схрещення визначає, які гени успадкує рішення наступної популяції відносно своїх предків. Операція мутації випадковим чином вибирає ген хромосоми та присвоює йому випадково вибрану фізичні машини. Мутації мають велике значення для методів що базуються на генетичних алгоритмах, оскільки це дає змогу значно збільшити різноманітність популяції. У наших експериментах для збільшення різновиду у популяції, вірогідність успадкування гену від обох предків дорівнює 0.5. Ми не присвоюємо більшу вірогідність успадкування гену предку із більшим значенням функції пристосованості.

Запропонований генетичний алгоритм у більшій мірі схожий на традиційні генетичні алгоритми. На початку роботи алгоритму випадковим чином створюється початкова популяція рішень-індивідів. Далі, на кожній ітерації алгоритму ми обчислюємо значення функції пристосованості кожного індивіду, для кожного індивіда у популяції вибирається пара для генерації індивідів наступної популяції. Після цього ми застосовуємо операцію мутації. Окрім того, ми виконуємо пошук найкращого індивіду нової популяції та порівнюємо його із найкращим індивідом попередньої популяції. Якщо найкращий індивід нової популяції кращий ніж найкращий індивід попередньої популяції, ми зберігаємо новий індивід як останній найкращий. У наступній ітерації алгоритму ми продовжуємо роботу із індивідами нової популяції, повністю відкидаючи попередню популяцію, оскільки індивіди нової популяції містять кращі рішення ніж індивіди попередньої популяції.

Розподіл ресурсів із врахуванням вимог високої доступності критичної IT-інфраструктури. До об'єктів критичної IT-інфраструктури пред'являють особливі вимоги щодо забезпечення високої доступності. Введемо наступні поняття:

– відмовостійкість – здатність системи до подальшої роботи після відмови одного із її елементів;

– неперервна доступність – здатність системи до безперервного обслуговування, незалежно від часу відмови вузлів системи;

– високодоступність – здатність системи до подальшої роботи після відмови одного із вузлів, з можливими перервами у роботі.

Як правило, при реалізації неперервної доступності виникає багато труднощів, подолання яких вимагає значних фінансових витрат. Водночас, існують ситуації, коли необхідно забезпечити відмовостійкість системи без жорстких вимог до неперервності доступу.

Розміщення елементів будь-якого високодоступного кластеру завжди повинне виконувати наступну вимогу: основні та резервні елементи критичної IT-інфраструктури повинні знаходитись на різних фізичних серверах. Алгоритми, що наведені у попередніх розділах, можуть бути просто модифіковані для уведення додаткових вимог щодо розміщення віртуальних машин шляхом впровадження додаткових "ресурсів".

У (15) ресурс є кумулятивним об'єктом: сумарне споживання ресурсу дорівнює сумі споживань цього ресурсу кожною віртуальною машиною. Для кожного елементу високодоступного кластеру, що повинен знаходитись на окремій фізичній машині, визначимо його "спільний" ресурс за (17):

$$u_j^c = \begin{cases} +\infty, |V_j \cap V^c| > 1, \\ 1, |V_j \cap V^c| \le 1, \end{cases} \quad (17)$$

де V^c – множина віртуальних машин, що належать кластеру c.

Таким чином, при виявленні на одній фізичній машині більш ніж одного елементу кластеру, у фітнес-функцію вводиться додаткова штрафна величина. Для більш загального випадку, коли необхідно виконати розміщення кластеру, що використовує певну мінімальну кількість фізичних серверів, визначимо ресурс за (18) та (19):

$$u_j^c = \begin{cases} +\infty, |V_j \cap V^c| > 1, \\ 1, |V_j \cap V^c| \le I, \end{cases} \quad (18)$$

$$I = C - k + 1, \quad (19)$$

де C – кількість елементів високодоступного кластеру;

k – мінімальна кількість різних фізичних машин, що повинні бути задіяні.

Експериментальні дослідження. Для порівняння вибрані та реалізовані декілька алгоритмів розміщення:

– Random Allocation – випадкове розміщення;

– First Fit Decreasing – перший підходящий за спаданням;

– Demand Risk – алгоритм запропонований у [17];

– CloudSim – алгоритм, що використовується у середовищі CloudSim – програмному засобі для симуляції роботи фізичних та віртуальних машин у хмарних середовищах.

Для експериментальних досліджень створено різні тестові сценарії для порівняння наведених алгоритмів розміщення.

Кількість поколінь генетичного алгоритму становить 20, розмір популяції кожного покоління становить 100. Розмір підпопуляції під час вибірки становить 5% від загального розміру популяції. Критерієм зупинки являється досягання алгоритмом 20-го покоління. Експериментально було досліджено, що у даному випадку більша кількість поколінь не приводить до значно кращого результату.

Отже, значення параметрів компонентів алгоритму є такими:

– у операції схрещення вірогідність наслідування гену від кожного предку дорівнює 0.5. Це означає, що у нащадка є однаковий шанс унаслідувати ген від обох предків. Як зазначено вище, це значення було підібрано для підвищення різновидності популяції;

– у операції вибірки розмір підпопуляції становить 5, що дорівнює 5% від розміру загальної популяції.

Загалом, критерієм зупинки генетичного алгоритму являється досягнення алгоритмом ітерації 20. У експериментах ми побачимо, що це значення є достатнім, оскільки застосування більшої кількості поколінь практично не змінює значення функції пристосованості кращого індивіда.

У генетичному алгоритмі вірогідність мутації дорівнює 0.02. Типові значення параметрів мутації зазвичай не перевищують 0.1, як видно у [18] та [19].

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). Також в усіх тестових сценаріях кількість фізичних машин фіксована та дорівнює 200, в той час як кількість сервісів варіюється між 200 та 1000 з кроком 200.

У першому сценарії параметри встановлено згідно табл. 3. Для порівняння алгоритмів визначено, як змінюється ступінь марнування ресурсу CPU зі збільшенням кількості сервісів. Результати дослідження наведені на рис. 7.

Таблиця 3 – Розподіл ресурсів сценарію 1

Ресурс	Сервіс	ФМ
CPU	250 Mips	2500 Mips
RAM	0.5 GB	5 GB
Network	0.1 Gbps	1 Gbps

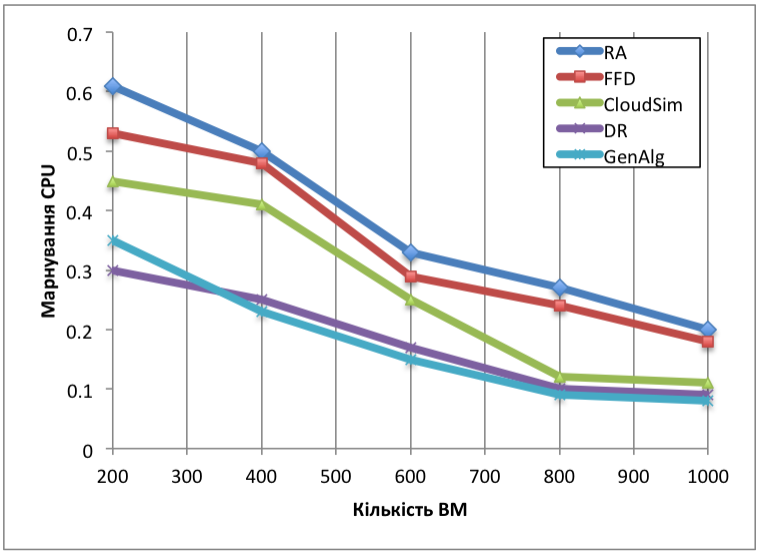

Рисунок 7 – Марнування ресурсу CPU у сценарії 1: алгоритми "Random Allocation", "First Fit Decreasing", "Demand Risk", "CloudSim" та "Genetic Algorithm"

У другому сценарії параметри були вибрані випадковим чином за допомогою рівномірного розподілу, межі значень яких наведені в табл. 4 та 5. Для порівняння алгоритмів, ми визначили, як змінюється кількість задіяних фізичних машин кожного алгоритму. Результати експерименту наведені на рис.8.

Таблиця 4 – 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 5 – Розподіл ресурсів ФМ, сценарій 2

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

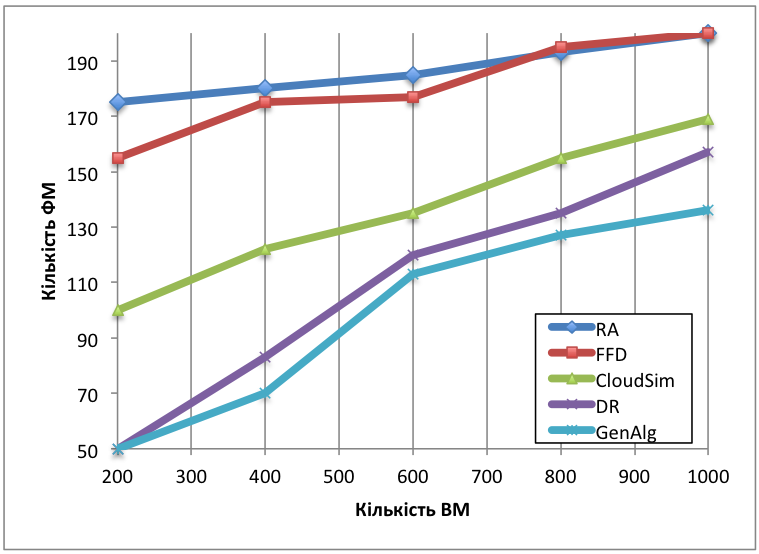

Рисунок 7 – Кількість використаних ФМ у сценарії 2: алгоритми “Random Allocation”, “First Fit Decreasing”, “Demand Risk”, “CloudSim” та “Genetic Algorithm”

Висновки. В роботі проведено аналіз основних хмарних провайдерів та визначено хмарну модель, яка в подальшому використана для реалізації моделі розподілу ресурсів критичної IT-інфраструктури з чіткими параметрами на базі генетичного алгоритму. В статті детально описано саму модель розподілу ресурсів, її використання для оптимізації функціонування критичних IT-інфраструктур. Проведені дослідження створеної моделі та наведені результати цих досліджень.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Л. С. Глоба, М. А. Скуліш, та О. М. Дяденко. *Математичні основи побудови інформаційно-телекомунікаційних систем*. Київ, Україна: Норіта-плюс, 2007.

[2] М. Горин, “Корпоративный ЦОД: за рамками технологий”, *Connect! Мир Связи*, № 8, с. 19-20, 2007.

[3] И. Кириллов, “Коммерческие ЦОД в Украине: новый этап развития”, *Сети и бизнес*, № 3 (52), 2010. [Електронний ресурс]. Доступно: http://www.sib.com.ua/arhiv2010/2010_3/statia_3_1_2010/statia_3_1_2010.htm.

[4] L. Badger, T. Grance, R. Patt-Corner, and J. Voas, “Cloud Computing Synopsis and Recommendations. Recommendations of the National Institute of Standards and Technology”, *Special Publication 800-146*. NIST, 2012.

[5] Н. Биберштейн, и С. Боуз, *Компас в мире сервис-ориентированной архитектуры (SOA)*. Москва, Российская Федерация: КУДИЦ-Пресс, 2007.

[6] D. Krafzik, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices*. Upper Saddle River, USA: Prentice Hall Professional, 2004.

[7] R. Kaur, and A. Kaur, "A Review Paper on Evolution of Cloud Computing, its Approaches and Comparison with Grid Computing", *International Journal of Computer Science and Information Technologies*, vol. 5, no. 5, pp. 6060-6063, 2014.
[8] A. Gupta, O. Sarood, and L. Kale, "Optimizing VM Placement for HPC in the Cloud", *International Letters of Social and Humanistic Sciences*, vol. 16, pp. 1-6, 2014. doi: 10.1145/2378975.2378977.
[9] J. Joseph, and C. Fellenstein, *Grid Computing*. Upper Saddle River, USA: Prentice Hall Professional, 2004.
[10] J. Nabrzyski, J. Schopf, and J. Węglarz, *Grid Resource Management: State of the Art and Future Trends*. Berlin, Germany: Springer, 2004.
[11] B. Goldworm, and A. Skamarock, *Blade servers and virtualization: transforming enterprise computing while cutting costs*. Hoboken, USA: Wiley Publishing, 2007.
[12] N. Ruest, and D. Ruest, *Virtualization, A Beginner's Guide*. New York, USA: McGraw Hill Professional, 2009.
[13] С. Ф. Теленик, О. І. Ролік, М. М. Букасов, та А. Ю. Лабунський, "Моделі управління віртуальними машинами при серверній віртуалізації", *Вісник НТУУ «КПІ»: Інформатика, управління та обчислювальна техніка*, № 51, с. 147-152, 2009.
[14] Дж. Холланд, "Генетические алгоритмы", *В мире науки*, № 9-10, с. 32-40, 1992.
[15] J. H. Holland, *Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence*. London, UK: Bradford book edition, 1992.
[16] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Boston, USA: Addison-Wesley, 1989.
[17] R. Buyya, J. Broberg, and A.M. Goscinski, *Cloud Computing: Principles and Paradigms*. New York, USA: John Wiley & Sons, 2010.
[18] R. Sookhtsaraei, M. Madani, and A. Kavian, "A multi objective virtual machine placement method for reduce operational costs in cloud computing by genetic", *International Journal of Computer Networks and Communications Security*, no. 8, с. 250-259, 2012.
[19] G. Wu, M. Tang, Y. Tian, and W. Li, "Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm", in *Proc. of International Conference on Neural Information Processing*, Lake Tahoe, 2012, pp. 315-323.

Стаття надійшла до редакції 24 вересня 2018 року.

REFERENCE

[1] L. S. Hloba, M. A. Skulish, ta O.M. Diadenko. *Mathematical Foundations of Construction of Information and Telecommunication Systems*. Kyiv, Ukraine: Norita-plius, 2007.
[2] M Gorin, "Corporate Data Center: Outside of Technology", Connect! The world of communication, no 8, pp. 19-20, 2007.
[3] I. Kirillov, "Commercial data centers in Ukraine: a new stage of development", *Networks and business*, no. 3 (52), 2010. [Online]. Available: http://www.sib.com.ua/arhiv2010/2010_3/statia_3_1_2010/statia_3_1_2010.htm.
[4] L. Badger, T. Grance, R. Patt-Corner, and J. Voas, "Cloud Computing Synopsis and Recommendations. Recommendations of the National Institute of Standards and Technology", *Special Publication 800-146*. NIST, 2012.
[5] N. Bibershtein, and S. Bouz, *Compass in the world of service-oriented architecture (SOA)*. Moskow, Russia: KUDITS-Press, 2007.
[6] D. Krafzik, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices*. Upper Saddle River, USA: Prentice Hall Professional, 2004.

[7] R. Kaur, and A. Kaur, "A Review Paper on Evolution of Cloud Computing, its Approaches and Comparison with Grid Computing", *International Journal of Computer Science and Information Technologies*, vol. 5, no. 5, pp. 6060-6063, 2014.
[8] A. Gupta, O. Sarood, and L. Kale, "Optimizing VM Placement for HPC in the Cloud", *International Letters of Social and Humanistic Sciences*, vol. 16, pp. 1-6, 2014. doi: 10.1145/2378975.2378977.
[9] J. Joseph, and C. Fellenstein, *Grid Computing*. Upper Saddle River, USA: Prentice Hall Professional, 2004.
[10] J. Nabrzyski, J. Schopf, and J. Węglarz, *Grid Resource Management: State of the Art and Future Trends*. Berlin, Germany: Springer, 2004.
[11] B. Goldworm, and A. Skamarock, *Blade servers and virtualization: transforming enterprise computing while cutting costs*. Hoboken, USA: Wiley Publishing, 2007.
[12] N. Ruest, and D. Ruest, *Virtualization, A Beginner's Guide*. New York, USA: McGraw Hill Professional, 2009.
[13] S. F. Telenyk, O. I. Rolik, M. M. Bukasov, A. Yu. Labunskyi,, "Models of virtual machine management with server virtualization"*, Bulletin of NTUU "KPI". Informatics, Management and Computer Science*, no. 51, pp. 147-152, 2009.
[14] D. Kholland, "Genetic Algorithms", *In the world of science*, no. 9-10, pp. 32-40, 1992.
[15] J. H. Holland, *Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence*. London, UK: Bradford book edition, 1992.
[16] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Boston, USA: Addison-Wesley, 1989.
[17] R. Buyya, J. Broberg, and A.M. Goscinski, *Cloud Computing: Principles and Paradigms*. New York, USA: John Wiley & Sons, 2010.
[18] R. Sookhtsaraei, M. Madani, and A. Kavian, "A multi objective virtual machine placement method for reduce operational costs in cloud computing by genetic"*, International Journal of Computer Networks and Communications Security*, no. 8, c. 250-259, 2012.
[19] G. Wu, M. Tang, Y. Tian, and W. Li, "Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm"*,* in *Proc. of International Conference on Neural Information Processing*, Lake Tahoe, 2012, pp. 315-323.

ЯРОСЛАВ ДОРОГИЙ,
ЕЛЕНА ДОРОГАЯ-ИВАНЮК,
ДМИТРИЙ ФЕРЕНС

МОДЕЛЬ РАСПРЕДЕЛЕНИЯ РЕСУРСОВ КРИТИЧЕСКОЙ ИТ-ИНФРАСТРУКТУРЫ С ЧЕТКИМИ ПАРАМЕТРАМИ НА БАЗЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Проведен детальный анализ исследований методов и алгоритмов размещения ресурсов виртуализированных ИТ-инфраструктур. Рассмотрена классическая модель облачных сервисов, которая состоит из трех слоев. Показано, что специфика задач, выполняемых в критических ИТ-инфраструктурах, ставит перед разработчиком повышенные требования по надежности, безопасности, доступности. Определено, что целесообразно использовать для реализации создаваемой модели сервис IaaS. Проанализированы основные поставщики данного облачного сервиса, определены их преимущества и недостатки, избран лучший кандидат для реализации. Далее приведено детальное описание математической модели распределения ресурсов критической ИТ-инфраструктуры с четкими параметрами и ее использование в сочетании с генетическим алгоритмом. Следующим в статье описано модель управления виртуальными машинами при серверной виртуализации. На примере показано, каким образом она используется для решения поставленной проблемы и каким образом ее можно оптимизировать и ускорить ее работу. В дальнейшем, в статье подробно

расписан генетический алгоритм, принцип построения фитнес-функции и его основные операции для решения поставленной задачи. Предложенный генетический алгоритм в большей степени похож на традиционные генетические алгоритмы. В начале работы алгоритма случайным образом создается начальная популяция решений-индивидов. Далее, на каждой итерации алгоритма вычисляется значение функции приспособленности каждого индивида, для каждого индивида в популяции выбирается пара для генерации индивидов следующей популяции. После этого применяется операция мутации. Кроме того, выполняется поиск наилучшего индивиду новой популяции и сравнивается с лучшим индивидом предыдущей популяции. И в завершение, для построенной модели приведен ряд уточнений, которые позволяют использовать данную модель для критической ИТ-инфраструктуры с учетом требований высокой доступности, таких как отказоустойчивость (способность системы к дальнейшей работе после отказа одного из ее элементов), непрерывная доступность (способность системы к непрерывному обслуживанию, независимо от времени отказа узлов системы) и высокодоступность (способность системы к дальнейшей работе после отказа одного из узлов, с возможными перерывами в работе). В последней части статьи приведены экспериментальные исследования предложенной модели распределения ресурсов критической ИТ-инфраструктуры с четкими параметрами на базе генетического алгоритма.

Ключевые слова: архитектура, облачные услуги, облачные сервисы, распределение ресурсов, генетический алгоритм, критическая ИТ-инфраструктура.

YAROSLAV DOROGYY,
OLENA DOROHA-IVANIUK,
DMYTRO FERENS

RESOURCES DISTRIBUTION MODEL OF CRITICAL IT INFRASTRUCTURE WITH CLEAR PARAMETERS BASED ON THE GENETIC ALGORITHM

The detailed analysis of researches of methods and algorithms of allocation of resources of virtualized IT-infrastructures is carried out. The classic model of cloud services, which consists of three layers, is considered. It is shown that the specificity of tasks performed in critical IT infrastructures puts the developer with increased requirements for reliability, security and availability. It is determined that it is expedient to use the service IaaS for implementation of the created model. The main providers of this cloud service were analyzed, their advantages and disadvantages were determined, the best candidate for implementation was selected. The following is a detailed description of the mathematical model of resource allocation of a critical IT infrastructure with clear parameters and its use in conjunction with the genetic algorithm. The following article describes the virtual machine management model for server virtualization. The example shows how it is used to solve the problem and how it can be optimized and accelerated. Subsequently, the article details the genetic algorithm, the principle of constructing a fitness function and its main operations to solve the problem. The proposed genetic algorithm is more similar to traditional genetic algorithms. At the beginning of the algorithm, an initial population of decision-individuals is created randomly. Next, each iteration of the algorithm calculates the value of the fitness function of each individual, for each individual in the population a couple is selected to generate individuals of the next population. After that, a mutation operation is applied. In addition, the search for the best individual of the new population is searched and compared with the best individual of the previous population. Finally, for the constructed model, a number of refinements are given that allow us to use this model for a critical IT infrastructure, taking into account high availability requirements such as fault tolerance (the ability of the system to continue working after the failure of one of its elements), continuous availability (the ability of the system to continuous maintenance, regardless of the time of failure of the system's nodes) and high availability (the ability of the system to further work after the failure of one of the nodes, with possible breaks in the work). The last part of the article presents experimental researches of the

proposed model of distribution of resources of critical IT infrastructure with clear parameters based on the genetic algorithm.

Keywords: architecture, cloud services, resource allocation, genetic algorithm, critical IT infrastructure.

Ярослав Юрійович Дорогий, кандидат технічних наук, доцент, доцент кафедри автоматики і управління в технічних системах, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна.

ORCID: 0000-0003-3848-9852.

E-mail: argusyk@gmail.com.

Олена Олександрівна Дорога-Іванюк, аспірантка, асистент кафедри автоматики і управління в технічних системах, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна.

ORCID: 0000-0003-3640-6312.

E-mail: dorogaya.helen@gmail.com.

Дмитро Андрійович Ференс, магістрант кафедри автоматики і управління в технічних системах, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна.

E-mail: ferensdima@gmail.com.

Ярослав Юрьевич Дорогой, кандидат технических наук, доцент, доцент кафедры автоматики и управления в технических системах, Национальный технический университет Украины "Киевский политехнический институт имени Игоря Сикорского", Киев, Украина.

Елена Александровна Дорогая-Иванюк, аспирантка, ассистент кафедры автоматики и управления в технических системах, Национальный технический университет Украины "Киевский политехнический институт имени Игоря Сикорского", Киев, Украина.

Дмитрий Андреевич Ференс, магистрант кафедры автоматики и управления в технических системах, Национальный технический университет Украины "Киевский политехнический институт имени Игоря Сикорского", Киев, Украина.

Yaroslav Dorohyi, candidate of technical sciences, associate professor, associate professor in Department of Automation and Control in Technical Systems, National technical university of Ukraine "Igor Sikorsky Kyiv polytechnic institute", Kyiv, Ukraine.

Olena Doroha-Ivaniuk, postgraduate student, assistant in Department of Automation and Control in Technical Systems, National technical university of Ukraine "Igor Sikorsky Kyiv polytechnic institute", Kyiv, Ukraine.

Dmytro Ferens, postgraduate student in Department of Automation and Control in Technical Systems, National technical university of Ukraine "Igor Sikorsky Kyiv polytechnic institute", Kyiv, Ukraine.