

Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Міністерство освіти і науки України

Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Міністерство освіти і науки України

Кваліфікаційна наукова  
праця на правах рукопису

**Омельченко Віталій Вікторович**

УДК 004

## **ДИСЕРТАЦІЯ**

### **Інформаційна технологія управління обчислювальними ресурсами в Kubernetes кластері**

126 – Інформаційні системи та технології

12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей,  
результатів і текстів інших авторів мають посилання на відповідне джерело

\_\_\_\_\_ В.В. Омельченко

Науковий керівник: Ролік О. І. , д.т.н., професор

Київ – 2025

## АНОТАЦІЯ

*Омельченко В. В.* Інформаційна технологія управління обчислювальними ресурсами в Kubernetes кластері. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 126 – Інформаційні системи та технології в галузі знань 12 – Інформаційні технології. – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2025.

Дисертаційна робота присвячена вирішенню проблеми автоматизації управління обчислювальними ресурсами в кластерах Kubernetes. Сучасні інформаційні системи підтримують велику кількість застосунків. Одним із важливих факторів для підтримання заданого рівня якості послуг є забезпечення компонентів інформаційної системи достатнім об'ємом обчислювальних ресурсів. Обчислювальні та фінансові ресурси є обмеженими, що вимагає постійного пошуку балансу між рівнем послуг та об'ємом ресурсів. Автоматизація процесів управління IT-інфраструктурою дозволяє підвищити ефективність використання обчислювальних ресурсів кластера при дотриманні необхідного рівня якості послуг, що надаються.

Проведено аналіз існуючих методів прогнозування часових рядів в контексті робочих навантажень на доцільність їх використання для проактивного масштабування. Для порівняння обрано поширені методи прогнозування для часових рядів з врахуванням сезонностей, тенденцій та запланованих подій. Проведені експериментальні дослідження оцінки точності обраних методів на типових шаблонах робочих навантажень на основі історичних даних різної довжини. Також досліджено стійкість даних методів до наявності аномалій в історичних даних. На основі проведених експериментів обрано множину методів для прогнозування робочих навантажень в проактивному масштабуванні.

Запропоновано метод комбінованого прогнозування з поєднанням компонентів довгострокового та короткострокового передбачення. Навантаження застосунків може включати аномалії, що призводить до некоректного розподілу обчислювальних ресурсів при проактивному управлінні. Метод комбінованого прогнозування дозволяє підвищити стійкість рішень для проактивного масштабування при наявності аномалій з визначеним шаблоном без постійної періодичності. Довгострокове прогнозування використовується для передбачення комплексних сезонностей і тенденцій. Короткострокове прогнозування відповідає за ідентифікацію аномалій в поточному навантаженні та визначення їх масштабу шляхом зіставлення поточного і минулих шаблонів. Комбінування прогнозів двох методів відбувається за допомогою вагових коефіцієнтів, які визначаються шляхом вирішення задачі максимізації на даних для тестування. Проведено експериментальне дослідження запропонованого методу. Результати демонструють підвищення загальної точності з 91% до 95% на тестових даних за умови наявності аномальних шаблонів в історичних даних. Використання запропонованого комбінованого методу прогнозування в рішеннях для проактивного масштабування може підвищити ефективність управління обчислювальними ресурсами.

Запропоновано метод проактивного масштабування обчислювальних ресурсів. Проактивні методи масштабування дозволяють надавати консистентний рівень якості послуг при ефективному використанні обчислювальних ресурсів. Описано модель проактивного управління. Запропоновано архітектуру модуля проактивного масштабування, яка включає модулі даних, прогнозування, застосування та прийняття рішень. Описано використання методу в умовах відсутності даних або низької точності отриманих прогнозів. На основі проведеного аналізу методів прогнозування обрано рішення для подальшого використання та розробки програмного модуля. Реалізовано програмний модуль для горизонтального проактивного масштабування в Kubernetes з використанням вбудованих

інструментів. Проведено експериментальне дослідження розробленого програмного модуля на кластері Kubernetes для порівняння зі статичним та реактивним підходами масштабування. У порівнянні з надлишковим статичним підходом отримано аналогічний рівень якості послуг при використанні на 46% меншого об'єму ресурсів. У порівнянні з реактивним підходом середній час відповіді застосунку зменшився з 160 мс до 23 мс.

Запропоновано гібридний метод масштабування, що включає реактивний і проактивний компоненти. Комбінація даних методів дозволяє використовувати проактивне управління при наявності точних прогнозів. При виникненні аномалій навантаження і неможливості точно оцінити їх масштаб, управління передається реактивному компоненту. Запропоновано індикатор переходу між проактивним і реактивним управлінням на основі порівняння відповідності отриманих прогнозів до поточного навантаження на застосунок. Рівень точності прогнозів визначається протягом заданої кількості ітерацій. Для забезпечення оперативної передачі управління забезпечується робота обох компонентів незалежно від поточного стану. Реалізовано програмний модуль з використанням розробленого проактивного методу і вбудованого рішення для реактивного масштабування в Kubernetes. Проведено експериментальне дослідження розробленого програмного модуля. Результати демонструють здатність запропонованого методу ідентифікувати аномалії та передавати управління між компонентами за визначеними правилами.

Запропоновано гібридний метод масштабування, що дозволяє координувати роботу горизонтального і вертикального компонентів масштабування. Метод дозволяє підвищити ефективність використання обчислювальних ресурсів кластера при використанні горизонтального масштабування. Вертикальний компонент адаптує об'єм обчислювальних ресурсів до поточних потреб в межах одного екземпляру для типів обчислювальних ресурсів, які не є основними при горизонтальному масштабуванні. Запропоновано модуль координації для узгодження

конфігурації вертикального і горизонтального компонентів. Розроблено програмний модуль для роботи в Kubernetes, який використовує запропонований метод. Результати проведених експериментів демонструють зменшення збиткового резервування обчислювальних ресурсів на 65% у порівнянні зі статичним підходом на синтетичних даних.

Створено інформаційну технологію управління обчислювальними ресурсами в кластері. Запропоновано здійснювати декомпозицію інформаційних систем на окремі модулі. Визначено функціонал кожного модуля і комунікацію між модулями та кластером. На основі запропонованої архітектури описано реалізацію інформаційної технології в кластерах Kubernetes. Запропоновано здійснювати інтеграцію вбудованих інструментів для моніторингу стану, аналізу даних і розгортання елементів управління. Описано процеси управління обчислювальними ресурсами. Реалізована інформаційна система використовує інтеграцію розроблених методів для проактивного та гібридного масштабування.

Розроблені методи та інформаційна технологія може бути використана для управління ресурсами в реальних інформаційних системах з використанням платформи Kubernetes або подібних платформ оркестрації контейнеризованих застосунків для автоматизації процесів управління обчислювальними ресурсами.

Ключові слова: інформаційні системи, інформаційні технології, віртуалізація, контейнеризація, Kubernetes, Docker, хмарні обчислення, управління ресурсами, мікросервісна архітектура, веб-застосунки, розподілені системи, математична модель, прогнозування, машинне навчання, програмне забезпечення.

## Список публікацій здобувача

*Наукові праці, в яких опубліковано основні наукові результати дисертації:*

1. Omelchenko V. Automation of resource management in information systems based on reactive vertical scaling / V.V. Omelchenko, O.I. Rolik // Adaptive systems of automatic control. – 2022. – Vol. 2, No. 41. – P. 65–78. – DOI: 10.20535/1560-8956.41.2022.271344.

2. Omechenko V. Integration of proactive and reactive approaches to scaling in Kubernetes / V.V. Omelchenko, O.I. Rolik // Scientific notes of Taurida National V.I. Vernadsky University. Series: Technical Sciences. – 2023. – No. 5. – P. 193–198. – DOI: 10.32782/2663-5941/2023.5/30.

3. Omechenko V. Proactive horizontal scaling method for Kubernetes / O.I. Rolik, V.V. Omelchenko // Radio Electronics, Computer Science, Control. – 2024. – No. 1. – P. 221–227. – DOI: 10.15588/1607-3274-2024-1-20.

*Наукові праці, які засвідчують апробацію матеріалів дисертації:*

1. Omelchenko V. Forecasting-at-scale algorithms for prediction cluster workload / V.V. Omelchenko, O.I. Rolik // The International Conference on Security, Fault Tolerance, Intelligence: тези доповідей Міжнар. конф. – Київ, 2023. – С. 1–5.

2. Omelchenko V. Workloads prediction methods for proactive resource scaling in Kubernetes / V.V. Omelchenko, O.I. Rolik // III International Scientific Symposium «Intelligent Solutions» (IntSol-23): тези доповідей Міжнар. конф. – Київ, 2023. – С. 44–53.

3. Omelchenko V. Combined forecasting method for cloud workloads / V.V. Omelchenko, O.I. Rolik // Problems of Infocommunications. Science and Technology (PIC S&T'2024): тези доповідей Міжнар. конф. – Харків, 2024.

## ABSTRACT

*Omelchenko V.* Information technology for computing resources management in Kubernetes cluster – Manuscript.

Thesis for the Doctor of Philosophy degree in the specialty 126 – Information systems and technologies in the knowledge field 12 – Information technologies. – National Technical University of Ukraine «Ihor Sikorsky Kyiv Polytechnic Institute», Kyiv, 2025.

The dissertation work is devoted to solving the problem of automating the management of computing resources in Kubernetes clusters. Modern information systems can include a large number of applications. One of the important factors in ensuring a given level of service quality is to provide the components of the information system with a sufficient amount of computing resources. Computing and financial resources are limited, which requires a constant search for a balance between the level of service quality and the amount of resources. Automation of management processes allows for an increase in the efficiency of using the cluster's computing resources while maintaining the required level of quality of services provided.

The existing forecasting methods are analyzed for the feasibility of their use for proactive scaling. For comparison, modern forecasting methods for time series with complex seasonality, trends, and planned events were selected. Experimental studies of the accuracy of the selected methods were conducted on typical workload templates based on historical data of various lengths. The stability of these methods to the presence of anomalies in historical data was also investigated. Based on the experiments, a set of methods was selected for predicting workloads in proactive scaling.

A hybrid forecasting method with a combination of long-term and short-term forecasting components is proposed. Application workloads may include anomalies, which leads to incorrect allocation of computing resources in proactive management. The combined forecasting method allows you to increase the stability of proactive

scaling solutions in the presence of anomalies with a defined pattern without a constant frequency. Long-term forecasting is used to predict complex seasonality and trends. Short-term forecasting is responsible for identifying anomalies in the current load and determining their scale by comparing current and past patterns. The forecasts of the two methods are combined using weighting coefficients, which are determined by solving a maximization problem on the test data. An experimental study of the proposed method was conducted. The results demonstrate an increase in overall accuracy from 91% to 95% on the test data, provided that there are anomalous patterns in the historical data. The use of the proposed combined forecasting method in proactive scaling solutions can improve the efficiency of computing resource management.

A method for proactive scaling of computing resources is proposed. Proactive scaling methods allow to provide a consistent level of QoS while efficiently using computing resources. A model of proactive management is described. An architecture that includes data, forecasting, application, and decision-making modules is proposed. An approach for the method to work in the absence of data or low accuracy of the obtained forecasts is described. Based on the analysis of forecasting methods, a solution was selected for further use and development of a software module. A software module for horizontal proactive scaling was implemented in Kubernetes using built-in tools. An experimental study of the developed software module on a Kubernetes cluster was conducted to compare it with static and reactive scaling approaches. Compared to the redundant static approach, a similar level of service quality was obtained using 46% less resources. Compared to the reactive approach, the average application response time decreased from 160 ms to 23 ms.

A hybrid scaling method is proposed that includes reactive and proactive components. The combination of these methods allows the use of proactive control in the presence of accurate forecasts. When load anomalies occur and it is impossible to accurately assess their scale, control is transferred to the reactive component. An indicator of the transition between proactive and reactive control is proposed based



on comparing the compliance of the obtained forecasts with the current load on the application. The level of accuracy of forecasts is determined within a given number of iterations. To ensure the prompt transfer of control, both components operate regardless of the current state. A software module was implemented using the developed proactive method and a built-in solution for reactive scaling in Kubernetes. An experimental study of the developed program module was carried out. The results demonstrate the ability of the proposed method to identify anomalies and transfer control between components according to certain rules.

A hybrid scaling method is proposed that allows coordinating the work of horizontal and vertical scaling components. The method makes it possible to increase the efficiency of using the cluster's computing resources when using horizontal scaling. The vertical component adapts the amount of computing resources to the current needs within a single instance for resource types that are not the main ones in horizontal scaling. A coordination module is proposed to coordinate the configuration of the vertical and horizontal components. Based on the proposed method, a program module for working in Kubernetes was developed. The results of the experiments demonstrate a 65% reduction in unprofitable reservation of computing resources compared to the static approach on synthetic data.

The information technology for managing computing resources in a cluster is described. The decomposition of information systems into separate modules is proposed. The functionality of each module and communication between modules and the cluster are described. Based on the proposed architecture, the implementation of information technology in Kubernetes clusters is described. The integration of built-in tools for status monitoring, data analysis, and deployment of controls is proposed. The management processes are described. The implemented information system integrates the developed methods for scaling.

Keywords: information systems, information technology, virtualization, containerization, Kubernetes, Docker, cloud computing, resource management, microservice architecture, web applications, distributed systems, mathematical model, forecasting, machine learning, software.

## ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ.....	13
ВСТУП .....	14
1 АНАЛІЗ АРХІТЕКТУРИ ІНФОРМАЦІЙНИХ СИСТЕМ З ВИКОРИСТАННЯМ КЛАСТЕРІВ УПРАВЛІННЯ РОЗПОДІЛЕНИМИ КОНТЕЙНЕРИЗОВАНИМИ ЗАСТОСУНКАМИ .....	18
1.1 Аналіз поточного стану і трендів керування ІТ-інфраструктурою..	18
1.2 Аналіз ІТ-інфраструктури та її елементів як об’єктів управління...	22
1.2.1 Загальна модель кластера .....	23
1.2.2 Структура кластера .....	25
1.2.3 Моніторинг кластера.....	28
1.3.4 Порівняльний аналіз віртуалізації і контейнеризації .....	30
1.3 Методи управління обчислювальними ресурсами .....	34
1.3.1 Аналіз методів управління обчислювальними ресурсами .....	35
1.3.2 Особливості горизонтального масштабування .....	39
1.3.3 Особливості вертикального масштабування .....	43
1.4 Аналіз задач управління обчислювальними ресурсами при управлінні якістю послуг .....	45
1.4.1 Обґрунтування необхідності дослідження та розробки методів масштабування.....	45
1.4.2 Визначення функціоналу рішень для управління ресурсами .....	49
Висновки до розділу 1 .....	51
2 РОЗРОБКА ПРОАКТИВНИХ МЕТОДІВ МАСШТАБУВАННЯ.....	52
2.1 Аналіз особливостей проактивного масштабування .....	53
2.2 Аналіз існуючих рішень для проактивного масштабування .....	54
2.3 Аналіз методів прогнозування .....	56
2.4 Розробка комбінованого методу прогнозування.....	65
2.4.1 Розробка методу комбінованого прогнозування.....	67
2.4.2 Дослідження комбінованого методу .....	71

2.5 Розробка проактивного методу масштабування .....	78
2.5.1 Сутність проактивного методу масштабування.....	78
2.5.2 Особливості реалізації проактивного методу масштабування .....	80
2.5.3 Дослідження проактивного методу .....	83
Висновки до розділу 2 .....	87
3 РОЗРОБКА ГІБРИДНИХ МЕТОДІВ МАСШТАБУВАННЯ .....	89
3.1 Метод проактивно-реактивного масштабування.....	89
3.1.1 Аналіз існуючих рішень для проактивно-реактивного масштабування.....	90
3.1.2 Сутність методу проактивно-реактивного масштабування.....	91
3.1.3 Дослідження методу проактивно-реактивного масштабування....	97
3.2 Метод горизонтально-вертикального масштабування.....	99
3.2.1 Аналіз існуючих методів інтеграції вертикального і горизонтального масштабування .....	101
3.2.2 Розробка координатора.....	102
3.2.3 Експериментальні дослідження горизонтально-вертикального методу .....	105
Висновки до розділу 3 .....	111
4 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ УПРАВЛІННЯ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ .....	112
4.1 Особливості обчислювальними ресурсами і навантаженням у кластері .....	112
4.1.1 Політики управління при перевищенні запитів на обчислювальні ресурси.....	112
4.1.2 Масштабування кластера.....	114
4.1.3 Балансування навантаження між екземплярами застосунків .....	115
4.2 Розробка інформаційної системи управління обчислювальними ресурсами .....	118
4.2.1 Розробка функціональної схеми інформаційної системи управління обчислювальними ресурсами.....	118

4.2.2 Функціональна схема модуля моніторингу .....	119
4.2.3 Функціональна схема модуля аналізу .....	119
4.2.4 Функціональна схема модуля прогнозування .....	120
4.2.5 Функціональна схема модуля планування.....	121
4.2.6 Функціональна схема модуля управління .....	121
4.5 Сутність інформаційної технології управління обчислювальними ресурсами .....	122
4.4 Особливості використання інформаційної технології управління обчислювальними ресурсами в Kubernetes .....	125
4.4.1 Особливості управління обчислювальними ресурсами в Kubernetes .....	125
4.4.2 Реалізація модулів інформаційної технології.....	126
Висновки до розділу 4 .....	135
ВИСНОВКИ.....	136
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	138
ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ В НАВЧАЛЬНИЙ ПРОЦЕС.....	149
ДОДАТОК Б. ЛІСТИНГ КОДУ КОМБІНОВАНОГО МЕТОДУ ПРОГНОЗУВАННЯ.....	150
ДОДАТОК В. ЛІСТИНГ КОДУ ПРОГРАМНОГО МОДУЛЮ ПРОАКТИВНОГО МАСШТАБУВАННЯ .....	152
ДОДАТОК Г. ЛІСТИНГ КОДУ ПРОГРАМНОГО МОДУЛЮ ГІБРИДНОГО МАСШТАБУВАННЯ .....	154

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

- ВМ – віртуальна машина
- ІС – інформаційна система
- ІТ – інформаційні технології
- ПЗ – програмне забезпечення
- СККЗ – системи керування контейнеризованими застосунками
- СУ – система управління
- СУІ – система управління інфраструктурою
- ОР – обчислювальні ресурси
- ОС – операційна система
- API – application programming interface – прикладний програмний інтерфейс
- CPU – central processing unit – центральний процесор
- ECS – elastic container service – сервіс контейнеризованих застосунків
- GKE – Google Kubernetes engine – платформа Kubernetes від Google
- HPA – horizontal pod autoscaler – горизонтальний масштабувач
- HTTP – hyper-text transfer protocol – протокол передачі гіпер-тексту
- IaaS – infrastructure as a service – інфраструктура як сервіс
- MAPE – mean absolute percentage error – середня абсолютне відхилення у відсотках
- OOM – out-of-memory – помилка нестачі пам'яті
- PM – pattern matching – зіставлення шаблонів
- PaaS – platform as a service – платформа як сервіс
- QoS – quality of service – якість послуг
- RAM – random access memory – пам'ять з довільним доступом
- RMSE – root mean square error – середньоквадратичне відхилення
- SLA – service-level agreement – угода про рівень послуг
- SLO – service-level objectives – цілі рівнів послуг
- SaaS – software as a service – програмне забезпечення як сервіс
- VPA – vertical pod autoscaler – вертикальний автоматичний масштабувач

## ВСТУП

**Актуальність теми.** В сучасному світі кількість державних і корпоративних послуг, що доступні через мережу Інтернет, збільшується кожного дня. Інформаційні технології (ІТ) застосовуються повсюдно і їх кількість також постійно збільшується. Забезпечення високої доступності та високого рівня QoS ІТ-послуг є ключовим аспектом для інформаційних систем (ІС), оскільки низький рівень QoS або відмова від обслуговування може призвести до критичних фінансових, репутаційних та ресурсних втрат [1]. Один із факторів забезпечення узгодженого рівня якості послуг є наявність необхідного об'єму обчислювальних ресурсів (ОР) ІТ-інфраструктури [2]. З іншого боку, обчислювальні та фінансові ресурси є обмеженими, через що виникає проблема підтримання рівня QoS при мінімізації об'єму ОР, що використовуються.

Сучасні кластери обробки даних можуть включати велику кількість різноманітних застосунків, кожен з яких має свої особливості та вимоги до ОР. Автоматизація процесів управління ОР є критично важливим аспектом функціонування інформаційних систем [3]. Поточні підходи до управління ОР часто є неефективними або з точки зору рівня QoS, або з точки зору рівня утилізації виділених ОР. Тому дана робота присвячена розробці ефективних методів автоматизації управління ОР для забезпечення заданого рівня QoS при мінімально необхідному обсязі ОР.

Зазвичай інформаційна система включає множину методів, програмних і технічних засобів, що об'єднані з метою автоматизації внутрішніх процесів, покращення їх стабільності та ефективності. У роботі вирішення задач управління ОР розглядається комплексно, тому пропонуються методи та засоби подолання існуючих проблем, які інтегруються задля забезпечення цілісного підходу до управління ОР, що включає моніторинг, аналіз даних, прогнозування, обробку помилок, координації та застосування конфігурацій ОР, що необхідні для роботи застосунків та включають процесорний час, обсяг

оперативної пам'яті, пропускну здатність каналів зв'язку, обсяги сховища даних та інші. В роботі управління ОР розглядається в контексті Kubernetes, його можливостей та обмежень, з декількох причин. По-перше, Kubernetes є найбільш масовою платформою для розміщення контейнеризованих застосунків і доступна у всіх основних хмарних провайдерів у вигляді PaaS рішення [4]. По-друге, модульність та гнучкість Kubernetes надає можливість замінити існуючі компоненти або розширювати їх функціонал за допомогою власних програмних модулів [5]. По-третє, екосистема даної платформа включає в себе готові компоненти для управління ОР, що спрощує подальші дослідження і дозволяє проводити порівняльний аналіз запропонованих рішень з існуючими.

**Об'єктом дослідження** є процес розподілу обчислювальних ресурсів для контейнеризованих застосунків в ІТ-інфраструктурі з використанням Kubernetes.

**Предметом дослідження** є методи автоматизації управління обчислювальних ресурсів.

**Мета** роботи полягає в забезпеченні підтримання на узгодженому рівні якості ІТ-сервісів, що надається застосунками користувачам, при мінімізації об'ємів виділених обчислювальних ресурсів. Для досягнення даної мети вирішені такі завдання:

- аналіз поточного стану проблеми та існуючих методів управління ОР в ІС;
- аналіз і порівняння існуючих методів прогнозування для використання в проактивних методах;
- удосконалення існуючих методів прогнозування в контексті здатності передбачати масштаб виникаючих аномалій;
- розробка проактивного методу масштабування;
- розробка комбінованого методу масштабування з використанням проактивного і реактивного компонентів;

– розробка методу поєднання горизонтального і вертикального масштабування.

### **Наукова новизна отриманих результатів:**

– запропоновано метод проактивного вертикального та горизонтального масштабування обчислювальних ресурсів, який відрізняється здатністю працювати з робочими навантаженнями, які містять комплексні сезонності та тенденції, що дозволяє зменшити використання обчислювальних ресурсів при дотриманні встановленого рівня якості послуг;

– запропоновано метод проактивно-реактивного масштабування, який відрізняється перерозподілом управлінням обчислювальними ресурсами між компонентами вертикального та горизонтального масштабування, що дозволяє забезпечувати встановлений рівень якості послуг у разі відсутності історичних даних, недостатньої точності отриманих прогнозів або некоректної роботи проактивного компонента;

– запропоновано метод горизонтально-вертикального масштабування, який відрізняється наявністю координатора для узгодження керуючих дій між компонентами масштабування, що дозволяє зменшити збиткове резервування обчислювальних ресурсів при використанні горизонтального масштабування для проактивного та реактивного управління;

– набула подальшого розвитку інформаційна технологія управління обчислювальними ресурсами ІТ-інфраструктури, яка відрізняється наявністю проактивного та гібридного масштабування, що дозволяє надавати консистентний рівень якості послуг.

**Практичне значення отриманих результатів.** Розроблені програмні модулі для проактивного масштабування можуть бути використані в кластерах Kubernetes для управління ОР застосунків, які мають чітку комплексну сезонність, тенденції та повторювані шаблони без сталої періодичності в навантаженні, для покращення рівня QoS у порівнянні з поширеними реактивними рішеннями та підвищення ефективності утилізації ОР у порівнянні зі статичним підходом. Запропонований метод комбінованого



прогнозування може бути використаний в ІТ-інфраструктурах, в яких часто виникають аномальні навантаження, що повторюються, але при цьому містять постійні сезонності та тенденції. Гібридний метод масштабування, що містить реактивний та проактивний компоненти, може бути використаний в ІТ-інфраструктурах з використанням Kubernetes для підвищення ефективності роботи обраного проактивного методу. Гібридний метод масштабування, що містить вертикальний та горизонтальний компоненти, можуть бути використані в існуючих ІТ-інфраструктурах для зменшення збиткового резервування ОР.

Результати досліджень, що включені до дисертаційної роботи, доповідались і обговорювались на конференціях:

- Міжнародна конференція «Security, Fault Tolerance, Intelligence», Київ, 29-30 липня, 2023;
- Третій міжнародний науковий симпозіум «Intelligent Solutions», Київ - Ужгород, 27-28 вересня, 2023;
- Міжнародна конференція «Problems of Infocommunications. Science and Technology», Харків, 5-7 жовтня, 2024.

Дослідження та розробка інформаційної технології проводились в рамках ініціативної науково-дослідницької роботи «Інтелектуальні високопродуктивні технології управління технічними системами» (державний реєстраційний номер – 0121U110810).

### **Структура та обсяг роботи.**

Дисертаційна робота складається із вступу, чотирьох розділів, висновків, списку використаних джерел (83 найменування) і чотирьох додатків. Основний зміст викладений на 125 сторінках друкованого тексту, містить 88 рисунків та 8 таблиць. Загальний обсяг дисертації – 154 сторінки.

# **1 АНАЛІЗ АРХІТЕКТУРИ ІНФОРМАЦІЙНИХ СИСТЕМ З ВИКОРИСТАННЯМ КЛАСТЕРІВ УПРАВЛІННЯ РОЗПОДІЛЕНИМИ КОНТЕЙНЕРИЗОВАНИМИ ЗАСТОСУНКАМИ**

## **1.1 Аналіз поточного стану і трендів керування ІТ-інфраструктурою**

Сучасні ІС надають користувачам широкий спектр різноманітних ІТ-послуг з різними значеннями показників доступності та надійності. Це вимагає декомпозиції монолітних застосунків на невеликі функціональні компоненти [6]. Забезпечення високої доступності ІТ-послуг вимагає дублювання компонентів, що їх надають, а також розміщення цих компонентів в різних географічних зонах [7]. Сучасні ІС підтримують велику кількість застосунків, кожен з яких розгортається в деякій кількості екземплярів. ІТ-інфраструктура, яка використовується для розгортання ІС, містить велику кількість фізичних серверів, на яких створюються віртуальні машини (ВМ) [4]. Управління великою кількістю серверів і застосунків, що розміщуються на них, є нетривіальною задачею і вимагає значного рівня автоматизації процесів управління, зокрема, розміщення та міграції ВМ, створення екземплярів застосунків, налаштування мережевого доступу до них, резервування необхідного для їх роботи об'єму ОР та моніторингу поточного стану [8]. Для управління ІТ-інфраструктурою розробляються та використовуються різноманітні системи управління, як комерційні, так і системи управління ІТ-інфраструктурою з відкритим вихідним кодом, до яких відносяться системи керування контейнеризованими застосунками (СККЗ) або системи оркестрації. Розробка і використання систем управління фактично призвели до виникнення повноцінного рівня між рівнем фізичної інфраструктури і рівнем сервісів – рівень платформи, що зображено на рис. 1.1 [9].

Основними задачами рівня платформи є розгортання і управління життєвим циклом застосунків. Для цього необхідно забезпечити роботу як з рівнем застосунків, так і з рівнем інфраструктури. Крім цього, на даному рівні відбувається формування запитів на об'єм ОР з метою підтримання якості

сервісів, що надаються застосунками на узгодженому рівні. Також, на рівні платформ реалізується процес моніторингу, який пов'язує низькорівневі інфраструктурні метрики з високорівневими метриками застосунків, що дозволяє в подальшому використовувати ці дані для автоматизації інших процесів ІС.

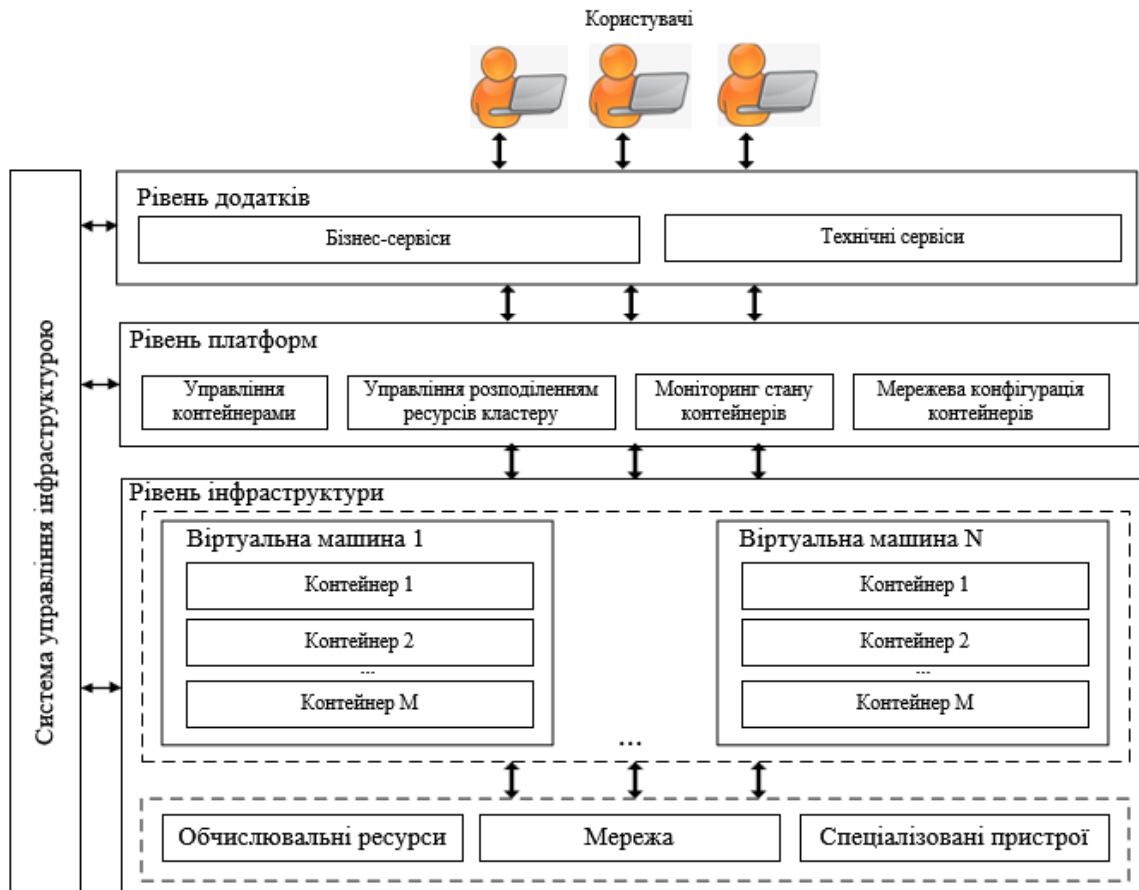


Рисунок 1.1 – Ієрархічна схема ІТ-інфраструктури

Використання при управлінні ресурсами ІТ-інфраструктури таких СККЗ як Kubernetes, Nomad та ECS [3] дозволяє значно спростити багато аспектів розгортання та керування застосунками в хмарних середовищах, а також в корпоративних ІТ-інфраструктурах. Для розгортання застосунків у кластері СККЗ забезпечує вищий рівень абстракції над апаратними засобами, віртуалізацією та ОС, що зображено на рис 1.2, що дозволяє ізолювати складність інфраструктури і уніфікувати взаємодію розробників застосунків з елементами інфраструктури.



Рисунок 1.2 – Ієрархічна схема ІТ-інфраструктури відносно оркестрації

СККЗ дозволяють автоматизувати процеси розгортання та масштабування завдяки вбудованим інструментам [10]. Надають можливість автоматичного додавання нових вузлів до кластера в автоматичному режимі, що включає налаштування моніторингу, мережевої та безпекової конфігурації, є критичним аспектом сучасної масштабованої ІС в контексті постійного масштабування [11].

СККЗ забезпечують високий рівень доступності шляхом відстежування збоїв у застосунках та їх автоматичне відновлення. Мережева конфігурація також включає балансування навантаження між екземплярами застосунків та налаштування внутрішніх з'єднань для компонентів ІС [12]. В контексті управління ОР, платформи для оркестрації дозволяють динамічно перерозподіляти об'єм виділених ОР для контейнерів, в яких розміщені екземпляри застосунків, шляхом внесення змін в конфігурацію запитів ОР або кількості екземплярів [6].

Кількість об'єктів управління в ІТ-інфраструктурах вимірюється тисячами, існує велика кількість невизначеностей стану усіх компонентів, компоненти можуть працювати в різних режимах, вирішуючи різноманітні задачі в умовах неповноти та недостовірності вихідних даних, наявності

різноманітних факторів ризику. За таких умов систему управління інфраструктурою (СУІ) доцільно проєктувати як ієрархічну дворівневу систему управління з координатором [1, 13, 14], що зображено на рис. 1.3.

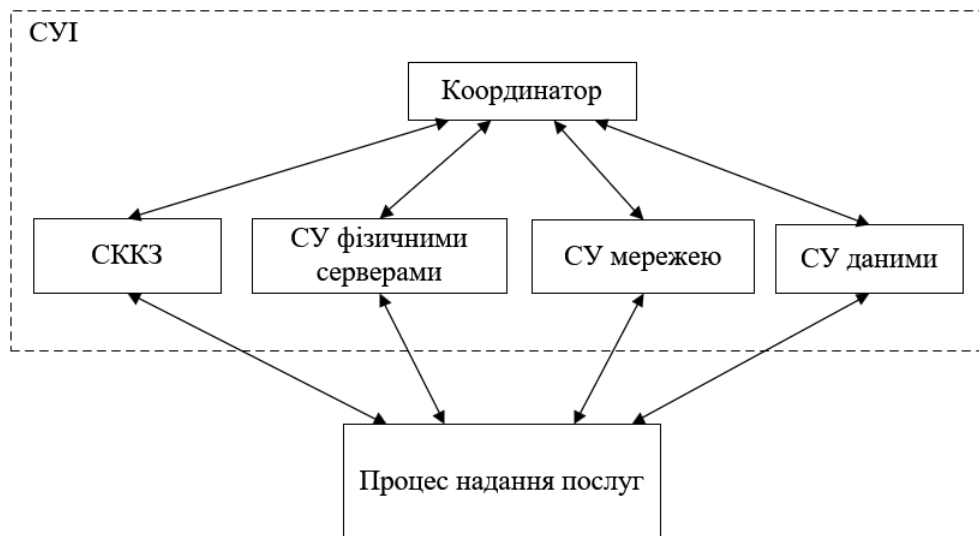


Рисунок 1.3 – Структура дворівневої СУІ

У такій моделі СУІ містить координатор, який розташований на верхньому рівні, та множину систем управління (СУ), кожна з яких вирішує окремі аспекти управління ІТ-інфраструктурою. Керуючі впливи координатора спрямовані на узгодження дій спеціалізованих СУ, що розташовані на нижньому рівні ієрархії. Кожна з цих СУ приймає рішення та керує об'єктами управління у своїй зоні відповідальності.

СУ фізичними серверами здійснює управління апаратними компонентами ІС та забезпечує конфігурування нових серверів для включення в ІС, моніторинг стану наявних серверів, оновлення їх програмного забезпечення (ПЗ) та відновлення у випадку неполадок. У випадку використання хмарних платформ для розміщення ІС СУ фізичними серверами керує ВМ.

СУ даними відповідає за розміщення, зберігання та захист даних в ІС. Ця СУ керує доступом до даних, що забезпечується шляхом визначення ролей і дозволів користувачів, використання відповідних протоколів аутентифікації.

Функціонал цих СУ містить організацію роботи реляційних та нереляційних баз даних, їх резервне копіювання та відновлення.

СУ мережею підтримує функціонування мережевої інфраструктури, її ефективну роботу, необхідний рівень безпеки та високу доступність. Управління мережевою інфраструктурою включає побудову топології мережі та її реконфігурацію для забезпечення безперебійної роботи компонентів ІС. Моніторинг мережі, в тому числі таких метрик, як пропускна здатність, затримка і втрата пакетів, є необхідним аспектом задля забезпечення високого рівня QoS. Використання віртуальних мереж підвищує рівень безпеки та дозволяє контролювання доступу в середині ІС.

Усі СУ, які наведено на рис. 1.3, а також інші СУ, які на рисунку не показано, разом з координатором відіграють важливу роль в підтриманні високого рівня якості сервісів, що надаються ІС. Для більшості з цих СУ напрацьовані методи та засоби управління, існують відповідні стандарти та специфікації. Менш опрацьовані питання управління контейнеризованими застосунками, які є зоною відповідальності СККЗ, через відносно малий термін від появи та широкого застосування таких технологій. Тому в роботі основна увага сконцентрована саме на управлінні обчислювальними ресурсами. СККЗ не тільки здійснює розміщення контейнеризованих застосунків на серверах ІТ-інфраструктури, а й керує масштабуванням ОР, виходячи з наявних обсягів ОР та з врахуванням вимог до якості послуг.

## **1.2 Аналіз ІТ-інфраструктури та її елементів як об'єктів управління**

Для з'ясування актуальних проблем управління ІТ-інфраструктурою, трендів та перспектив розвитку ІТ-інфраструктури на базі контейнеризації необхідно проаналізувати поточний стан, архітектуру, базову модель та основні концепції побудови ІТ-інфраструктури та її управління. Для ефективної роботи ІС необхідно забезпечувати скоординовану роботу всіх рівнів інфраструктури, технічних та бізнес-сервісів [1]. Така координація забезпечується шляхом використання інтегрованою СУІ, невід'ємною

частиною якої є СККЗ. СУІ використовує високий рівень абстракції для розгортання сервісів та підтримки необхідного рівня QoS.

Для підтримки високого рівня якості послуг СККЗ має виконувати такі функції [6]:

- забезпечувати надання необхідної кількості ОР для роботи розміщених застосунків;
- розміщувати застосунки щільно для максимізації утилізації доступних ОР;
- забезпечувати стійкість до відмов окремих компонентів або вузлів;
- гарантувати, що екземпляри застосунків є ізольованими і не впливають на роботу інших в межах одного вузла;
- обробляти вхідні запити або задачі та розподіляти їх між всіма екземплярами застосунку відповідно до встановлених політик балансування;
- здійснювати горизонтальне масштабування для адаптації до поточних вимог застосунків – додавати або видаляти вузли у разі нестачі ресурсів або їх неповної утилізації, відповідно.

Множину віртуальних або фізичних машин, які об'єднані спеціалізованим ПЗ в єдину систему називають кластером. Кластери використовуються для ефективного розподілу доступних ОР між застосунками, що дозволяє виконувати велику кількість різнотипних задач одночасно.

### 1.2.1 Загальна модель кластера

Для формулювання задачі розміщення застосунків у кластері необхідно формалізувати всі параметри, які будуть використовуватися при побудові моделі СККЗ.

Основу кластера складає множина  $N = \{N_i\}$ , де  $i = \overline{1, n}$  вузлів на фізичних або віртуальних серверах в залежності від типу інфраструктури, кількість яких дорівнює  $n$ .

В кластері розгортаються застосунки із множини  $A = \{A_l\}$ ,  $l = \overline{1, m}$ , де  $m$  – кількість застосунків. Кожен застосунок включає множину екземплярів  $I = \{I_{lj}\}$ ,  $l = \overline{1, m}$ ,  $j = \overline{1, k}$ , де  $k$  – кількість екземплярів застосунку  $I_l$ .

СККЗ керує ресурсами із множини  $R = \{R_y\}$ ,  $y = \overline{1, h}$ , де  $h$  – кількість типів ресурсів для управління. Такими ресурсами зазвичай є – процесорний час, оперативна пам'ять, пропускна здатність мережі, ємність дисків або наявність спеціальних пристроїв, зокрема, графічних процесорів та ін.

Кожен  $i$ -й вузол кластера має заданий об'єм ОР  $\{V_{yi}\}$ ,  $y = \overline{1, h}$ ,  $i = \overline{1, n}$ , де  $y$  – тип ОР. Дана характеристика визначає потужність  $i$ -го вузла.

Кожен  $l$ -й застосунок має вимоги до ОР  $\{X_{ly}\}$ ,  $y = \overline{1, h}$ , де  $h$  – кількість типів ресурсів для управління, які будуть надані кожному екземпляру  $I_{lj}$ ,  $l = \overline{1, m}$ ,  $j = \overline{1, k}$ , де  $k$  – кількість екземплярів застосунку  $I_l$ , при розміщенні на вузлах кластера. Фактичне використання ОР застосунків визначається множиною  $\{\hat{X}_{ly}\}$ , де може відрізнятися від встановлених вимог  $\{X_{ly}\}$  в залежності від поточного навантаження.

Головну мету управління ОР можна формалізувати у такий спосіб:

$$\begin{cases} Q_l - \hat{Q}_l \rightarrow 0 \\ \frac{\sum X_{ly}}{\sum \hat{X}_{ly}} \rightarrow 1 \end{cases}, l = \overline{1, m}, y = \overline{1, h}, \quad (1.1)$$

де  $Q_l$  та  $\hat{Q}_l$  – фактичне і цільове значення метрики якості послуг для  $l$ -го застосунку,  $l = \overline{1, m}$ , де  $m$  – кількість застосунків у кластері.

Таким чином, мета СККЗ при управлінні ОР полягає в підтриманні встановленого рівня якості послуг при мінімізації збиткового резервування ОР, як це визначається рівнянням (1.1).



СККЗ має забезпечувати для будь-якого екземпляра  $I_l$ , наявність на вузлі розміщення достатньої кількості ОР  $R_y$  для коректної роботи застосунку і виконання системи рівнянь (1.1):

$$\sum X_{lyi} \leq V_{yi}, l = \overline{1, m}, y = \overline{1, h}, i = \overline{1, n}, \quad (1.2)$$

де  $X_{lyi}$  – вимоги екземпляра  $l$ -го застосунку до ОР для  $y$ -го типу ресурсу, що розміщується на  $i$ -тому вузлі;  $V_{yi}$  – об’єм ОР доступний для  $i$ -го вузла для  $y$ -го типу ресурсу.

Таким чином, нерівність (1.2) визначає залежність між вимогами розміщених екземплярів на вузлах і їх потужністю, а саме, що сумарна потужність всіх екземплярів, розміщених на  $i$ -му вузлі, не може перевищувати його потужність. Підтримання даної нерівності гарантує, що вузли кластера будуть мати достатню кількість ресурсів для штатної роботи всіх розміщених на фізичному сервері застосунків.

### 1.2.2 Структура кластера

Структура СККЗ наведена на рис. 1.4. Управління кластером здійснює СККЗ. Для конфігурації ресурсів з боку адміністратора та розміщення застосунків, СККЗ містить інтерфейс для взаємодії з адміністратором. Модуль розміщення контейнерів здійснює пошук вузла для розміщення контейнера, використовуючи дані модуля моніторингу контейнерів. Взаємодія з фізичними або віртуальними машинами відбувається з використанням модуля взаємодії з вузлами, який безпосередньо розміщує застосунки в контейнерах на ВМ.

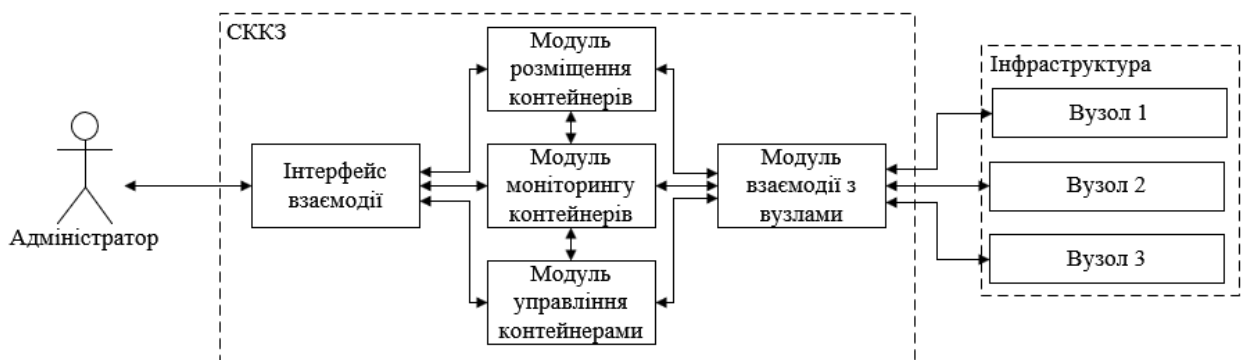


Рисунок 1.4 – Структура СККЗ

Модуль управління контейнерами на основі даних модуля моніторингу та команд управління, отриманих від адміністраторів кластера, приймає рішення про переміщення, перестворення чи видалення контейнерів. Модуль моніторингу контейнерів з використанням модуля взаємодії з вузлами збирає дані про поточний стан контейнерів у вигляді системних метрик.

На основі запропонованої структури СККЗ можна визначити архітектуру кластера. Кластер має складатися з одного або множини вузлів, які є фізичними або віртуальними машинами. На вузлах розміщуються контейнеризовані застосунки та компоненти управління кластером. Для забезпечення взаємодії вузла з іншими компонентами управління в кластері встановлюється агент [15].

Агент на рис. 1.5 є програмним модулем, який необхідний для виконання команд управління від СККЗ на розгортання, моніторинг та видалення контейнерів. Також, агент може відповідати за мережеву конфігурацію вузла для забезпечення з'єднання між розміщеними контейнерами та іншими компонентами кластера.

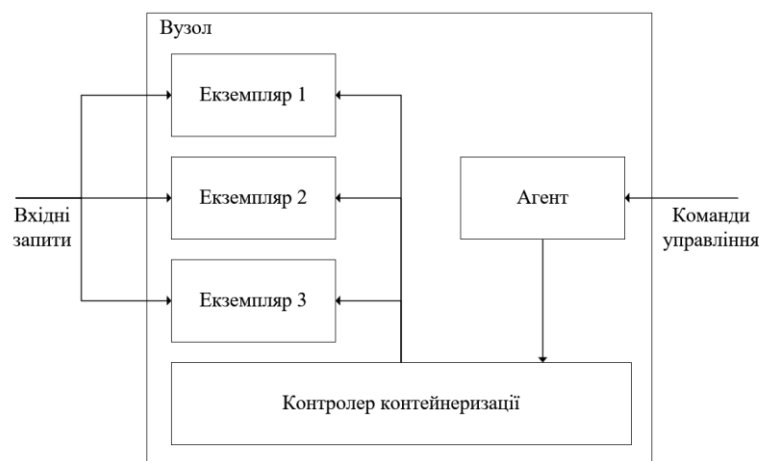


Рисунок 1.5 – Архітектура вузла кластера

При отриманні команд управління на розгортання нового контейнеру, агент передає дані про мережеву конфігурацію та конфігурацію ресурсів до СККЗ [16]. СККЗ за допомогою доступних на цьому вузлі інструментів операційної системи (ОС) запускає контейнер із заданою конфігурацією забезпечуючи вказаний об'єм ОР та мережевих ресурсів. СККЗ здійснює

моніторинг стану розгорнутих контейнерів та об'єму ОР, що використовуються. У разі перевищення обсягів ресурсів, що визначені при конфігуруванні ОР контейнер зупиняється, а агент отримує інформацію про подію, яку передає в СККЗ.

Кластер надає прикладний програмний інтерфейс (API) для конфігурування ресурсів кластера, що відповідає інтерфейсу взаємодії СККЗ. Адміністратори можуть створювати, оновлювати та видаляти конфігурації ресурсів шляхом надсилання команд на API.

База даних необхідна для обміну даними та командами управління між компонентами СУ та є частиною реалізації модулів СККЗ. Контролер застосунків, що відповідає модулю управління контейнерами, отримує дані про поточну конфігурацію застосунків і виконує відповідні операції для приведення поточного стану до вказаного [17]. Якщо контролер застосунків приймає рішення про розміщення нових екземплярів в кластері, то через збереження відповідної задачі в базі даних передає запит до планувальника.

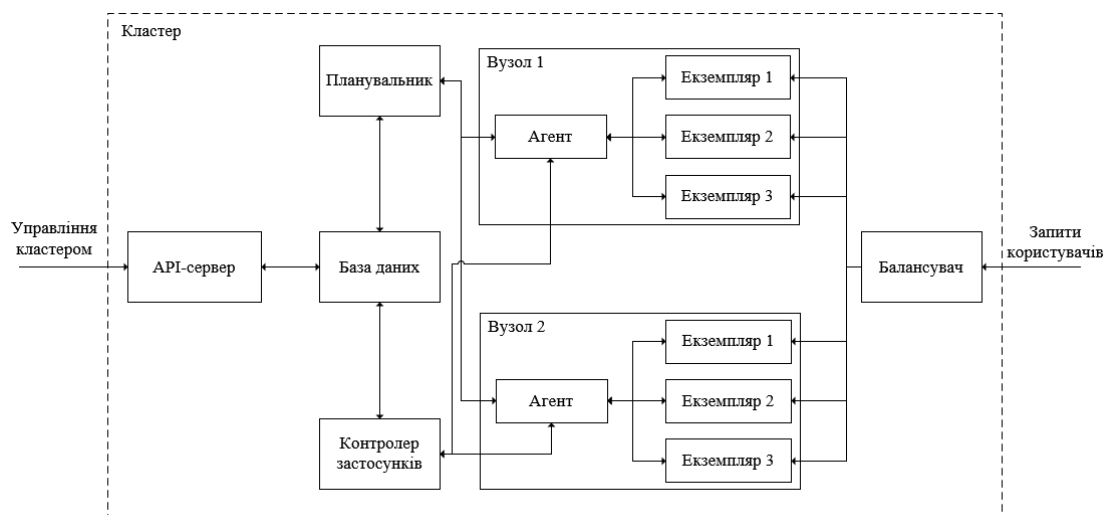


Рисунок 1.6 – Архітектура кластера

Планувальник розміщує нові екземпляри на доступних вузлах. Після знаходження вузла для розміщення, планувальник звертається до агента, який виконує відповідну операцію. Після розміщення екземпляру результат операції зберігається в базі даних. Планувальник є частиною реалізації модуля розміщення застосунків в структурі СККЗ.

### 1.2.3 Моніторинг кластера

Планування використання ОР вимагає отримання актуальних знань про поточний стан ІС. Сучасні ІС можуть включати тисячі вузлів, на кожному з яких може знаходитися сотні контейнеризованих програм. Тому для ефективного управління ОР в кластері необхідне рішення для збору даних про поточний стан компонентів та кластера в цілому – модуль моніторингу. Моніторинг є систематичним процесом збору, аналізу та використання інформації про стан та ефективність внутрішніх сервісів та розміщених застосунків у кластері. Цей процес включає відстеження різноманітних метрик і логів, які можуть надавати дані про поточну продуктивність, доступність та безпеку системи. Основна ціль моніторингу – визначати рівень QoS та перевіряти дотримання встановлених вимог SLA [18].

В Kubernetes метрика – це кількісний показник, який описує стан компонента в деякий момент часу в одному вимірі. Метрики можна розділити на системні і бізнес-метрики. Системні метрики пов'язані з роботою інфраструктури, а саме апаратної частини та ОС. Типовий перелік системних метрик:

- метрики утилізації ОР надає можливість оцінити використання ресурсу CPU, RAM та інших ресурсів кожним із розміщених контейнерів на вузлі, а також об'єм вільних ресурсів;
- метрики мережевої активності дозволяють оцінити поточне використання мережевого каналу, об'єм вхідного і вихідного трафіку;
- метрика I/O надає інформацію про кількість операцій вводу-виводу та їх швидкість;
- метрика помилок дозволяє проаналізувати наявність критичних ситуацій в роботі ядра ОС або апаратного забезпечення.

Системні метрики є базовим для ідентифікації проблем у роботі вузла або застосунків. Крім того, дані метрики є ключовими для оцінки ефективності використання ОР вузла.

Бізнес-метрики є більш високорівневими і відображають стан компонентів з точки зору досягнення бізнес-цілей. Перелік типових метрик даного виду:

- метрика часу відгуку надає інформацію про швидкість обробки запиту, яка часто є головним показником QoS;
- метрика пропускної здатності дозволяє оцінити кількість оброблених запитів за одиницю часу;
- метрика кількості транзакцій або подібні є високорівневими даними, які можуть збиратися з застосунків.

В процесі моніторингу здійснюється збір метрик та логів з застосунків і вузлів. Зібрані дані мають бути агреговані та збережені в централізоване сховище для їх подальшого використання автоматизованими компонентами СУІ або персоналом. Модуль моніторингу сповіщає про аномальні або критичні ситуації, наприклад, аварійне завершення екземпляру або різке падіння мережевого трафіку. Візуалізація є важливою частиною моніторингу, оскільки перетворює зібрані дані у формат, який зрозумілий для користувачів кластера. У разі наявності відповідних інструкцій, модуль моніторингу запускає автоматичні процеси відновлення.



Рисунок 1.7 – Основні задачі моніторингу

Для модуля моніторингу критичним атрибутом є здатність масштабуватися у контексті даних. Оскільки ІС можуть включати велику кількість компонентів, а кожний з них є потенційним джерелом даних про стан системи, тому модуль моніторингу має збирати, передавати та аналізувати великий обсяг даних. Більшість методів моніторингу базуються на принципі розподілення, коли агенти на кожному вузлі відповідають за попередній збір, фільтрацію і агрегацію даних [19, 20].

#### 1.3.4 Порівняльний аналіз віртуалізації і контейнеризації

Технологія віртуалізації повсюдно використовується при розробці сучасних ІС. Фактично ІaaS повністю залежить від даної технології, оскільки хмарні провайдери надають клієнтам в користування саме розгорнуті на їх фізичних серверах віртуальні машини. Крім того, РaaS та SaaS будуються на основі ІaaS, що означає, що всі застосунки, бази даних та інші сервіси також запускаються за допомогою технології віртуалізації [21].

Віртуалізація – це технологія створення додаткового рівня абстракції між фізичними ОР та їх логічною репрезентацією в штучному середовищі. Дана технологія дозволяє створювати ВМ, які поводять себе як повноцінні фізичні сервери. Кожна така ВМ має власну ОС та віртуальні ОР, що дозволяє запускати декілька середовищ виконання програм на одному фізичному сервері.

Віртуалізація дозволяє значно підвищити ефективність використання ОР, знизити фінансові витрати на електроенергію та спростити обслуговування серверів. Кожна ВМ ізольована від інших ВМ, що розміщена на одному фізичному сервері. Це забезпечується шляхом обмеження доступу до пам'яті та мережевих інтерфейсів, а також створенням власної файлової системи. Віртуалізація значно підвищує рівень безпеки [22], оскільки шкідливе ПЗ на одній ВМ не може впливати на роботу інших ВМ засобами ОС.

Іншою важливою перевагою технології віртуалізації є можливість швидко розгортати нові навантаження, оскільки ВМ швидко розгортаються на

існуючих фізичних серверах, можуть бути переміщені разом з критичними даними у разі необхідності або видалені. Віртуалізація дає можливість розробникам ПЗ отримувати уніфіковану платформу для розробки, оскільки дана технологія дозволяє емулювати процесорну архітектуру ВМ без залежності до архітектури фізичного сервера.

В контексті управління ОР, віртуалізація дозволяє обмежувати використання ресурсів сервера та динамічно їх змінювати у разі потреби без перезавантаження розміщеного застосунку при наявності доступних ОР на фізичній машині.

Незважаючи на значні переваги віртуалізації, дана технологія має також недоліки і обмеження. Зокрема, наявність додаткового рівня абстракції знижує загальну продуктивність у деяких типах віртуалізації. Цей недолік може бути критичним для ресурсоемних застосунків, що вимагають швидкої обробки даних або вводу-виводу [23].

Незважаючи на те, що віртуалізації надає абстракцію над фізичним обладнанням, ВМ все ще є залежними від підлеглих платформ та їхніх можливостей. Проблеми на рівні фізичного сервера можуть вплинути на роботу розміщених на ньому ВМ. Крім того, при міграції ВМ на інші фізичні сервери можна зіштовхнутися з проблемами несумісності та необхідності адаптації образів. Незважаючи на високий рівень ізоляції та безпеки, атаки на гіпервізор або помилки в його реалізації можуть призвести до отримання доступу до інших ВМ небажаними особами.

Існує кілька методів віртуалізації, класифікацію яких наведено на рис. 1.8. Віртуалізація на рівні ОС, або контейнеризація, забезпечує ізолюваність запущених застосунків шляхом використання спеціалізованих вбудованих інструментів ОС. Цей метод не вимагає емуляції або віртуалізації окремих фізичних ресурсів. Натомість процеси ізолюються за допомогою простору імен та керуванням ресурсами на рівні ОС, зокрема `cgroups` [24]. Прикладами реалізації даного методу є `Docker` та `containerd` [25].

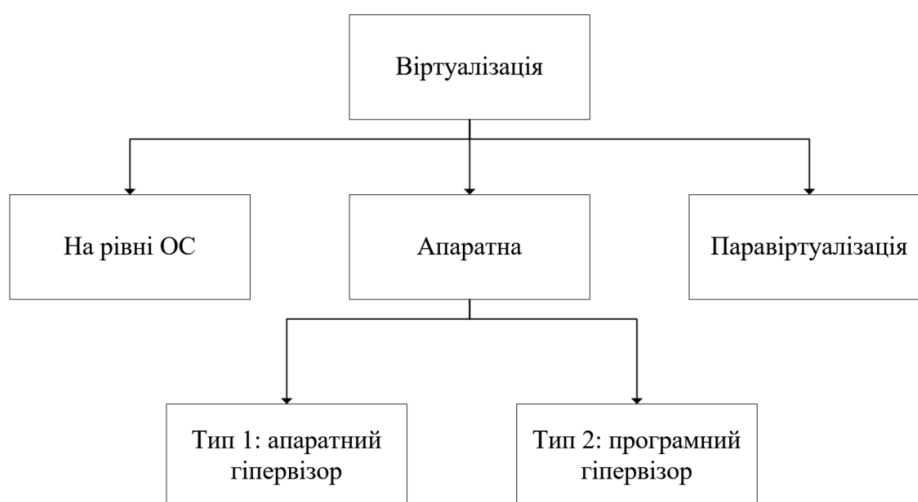


Рисунок 1.8 – Класифікація методів віртуалізації

Паравіртуалізація також використовує хостову ОС, але при цьому вимагається модифікація ядра для використання спеціалізованого API. Даний метод дозволяє знизити накладні витрати, пов'язані з емуляцією апаратного забезпечення, що дозволяє збільшити продуктивність у порівнянні з методом апаратної віртуалізації. Прикладом реалізації даного методу є Xen [26].

Апаратна віртуалізація повністю емує апаратне забезпечення, що, з одного боку, надає найкращий рівень ізоляції, але з іншого призводить до значних втрат продуктивності. Апаратна віртуалізація може бути двох типів.

При першому типі використовується апаратний гіпервізор, який надає високий рівень продуктивності завдяки прямій взаємодії з CPU, RAM та пристроями вводу-виводу, що прибирає накладні витрати комунікації з ОС. Проте перший тип віртуалізації вимагає наявності відповідного апаратного забезпечення.

Віртуалізація другого типу використовує програмний гіпервізор. Гіпервізори другого типу інтегровані в ОС та мають нижчий рівень продуктивності у порівнянні з апаратною віртуалізацією через наявність додаткової комунікації на рівні ОС. На рис. 1.9 зображено порівняння архітектури типів віртуалізації з гіпервізором і без нього.



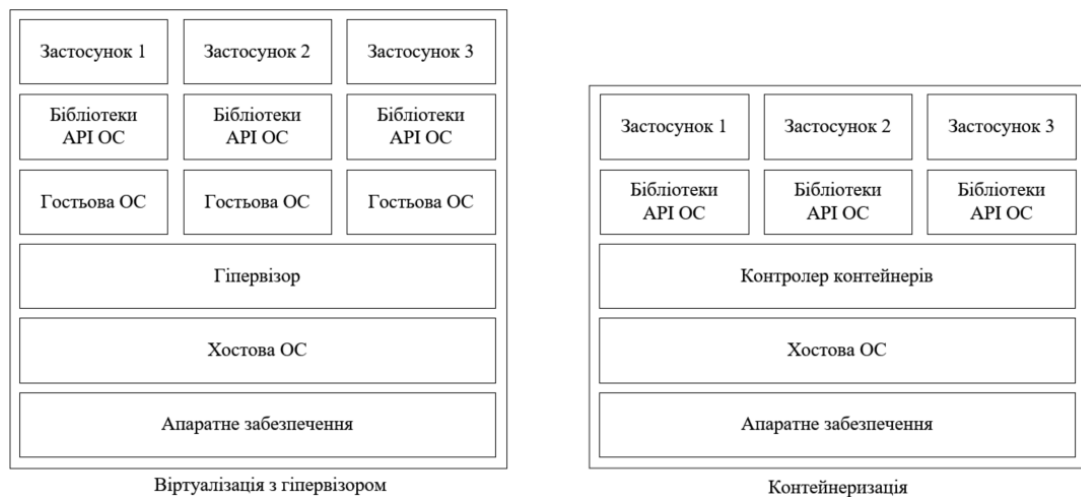


Рисунок 1.9 – Порівняння архітектур віртуалізації і контейнеризації

В СККЗ доцільно використовувати віртуалізацію рівня ОС через низку причин. Зокрема, контейнеризація має кращу швидкодію, менші накладні витрати та швидше розгортається, що є критичними факторами для ІС. Абсолютна більшість контейнерів базується на використанні ОС Linux [27].

Програми, що виконуються в контейнерах Linux, є звичайними процесами і спільно використовують ядро ОС. Такі процеси виконуються в окремих просторах імен ядра ОС, що не дозволяє будь-якому процесу отримати доступ до інших процесів. Крім того, кожна контейнеризована програма має власний простір файлової системи, подібно до виклику операції `chroot` в Unix-подібних ОС. Для ізоляції ресурсів використовується механізм груп керування `groups`. Одним із недоліків даного механізму є те, що процеси всередині контейнера не мають інформації про обмеження ОР – процес може бачити всю доступну RAM, але при цьому мати доступ лише до її частини. Іншим недоліком є неможливість змінити встановлені ліміти ОР без завершення роботи процесу, що ускладнює ІТ управління контейнеризованими застосунками.

В роботі [28] автори дослідили продуктивність виконання Spark задач з використанням контейнеризації і віртуалізації, з порівнянням часу виконання паралельних типових Spark задач. Результати експериментів показали, що при зростанні кількості контейнерів крива часу виконання зростає значно

повільніше. Крім того, на одному фізичному сервері можна розмістити значно більше контейнерів ніж VM. В іншому експерименті досліджувалося використання CPU під час проведення попереднього дослідження. Зроблено висновок, що при виконанні еквівалентних задач контейнеризація потребує значно менше процесорного часу.

Через знижені накладні витрати у порівнянні з VM, кращу продуктивність і наявність відповідного рівня ізоляції забезпечує контейнеризація. Тому вона є превалюючим інструментом при побудові ІС. Швидкість розгортання VM напряду впливає на можливість динамічного масштабування застосунків, що є одним із ключових факторів надання необхідного рівня QoS та виконання умов (1.1).

### **1.3 Методи управління обчислювальними ресурсами**

Управління ресурсами в ІТ є головним аспектом для забезпечення ефективності, стабільності та масштабованості ІС у складних та динамічних середовищах. В умовах, коли бізнес-процеси тісно пов'язані з ІТ-інфраструктурою, відповідне керування ОР є вирішальним фактором для досягнення стратегічних бізнес-завдань [1].

Управління ресурсами в ІС є комплексним процесом, який включає планування, розподіл та використання наявних ОР задля забезпечення ефективної роботи ІС в цілому та надання інформаційних послуг користувачам з виконанням вимог (1.1).

Управління ОР відіграє ключову роль у дотриманні вимог SLA. Наприклад, ефективне управління ресурсами може гарантувати, що під час пікових навантажень застосунки будуть належним чином масштабовані для підтримки високого рівня доступності. Водночас, у періоди низького навантаження, система масштабується вниз, зменшуючи витрати, але при цьому залишаючись у межах показників QoS, що зафіксовані в SLA. Некоректне управління ресурсами як і його відсутність призводить до порушень вимог SLA, фінансових та репутаційних втрат.

Сучасні ІС є багатофункціональними і можуть надавати сотні різних сервісів користувачам, що досягається декомпозицією монолітних застосунків на компоненти в рамках мікросервісної архітектури. Це, в свою чергу, означає, що ІС можуть містити сотні пов'язаних між собою застосунків з унікальними особливостями роботи і потребами для кожного з них. Для управління такими ІС, а також можливістю масштабуватися в майбутньому, процеси мають бути автоматизованими, оскільки людський фактор може значно обмежувати та сповільнювати процеси управління. Тому в роботі вирішуються задачі автоматизації процесів управління ОР.

### 1.3.1 Аналіз методів управління обчислювальними ресурсами

Для забезпечення дотримання SLA і підтримки високого рівня QoS в ІС, СККЗ, зокрема Kubernetes, надають відповідні інструменти, які дозволяються гнучко керувати основними аспектами розподілення і використання потужностей кластера [29].



Рисунок 1.10 – Інструменти управління ОР

Масштабування є процесом збільшення або зменшення об'єму ресурсів наданого застосунку або кластера задля оптимізації витрат та підтримки необхідного рівня QoS. Масштабування застосунків відбувається завдяки використанню ресурсів кластера. Якщо об'єм ресурсів кластера недостатній або утилізується не повністю, кластер також може бути масштабований. Масштабування є головним інструментом автоматизації управління ресурсами у кластері через свою гнучкість.

Зміна кількості екземплярів застосунку визначається як горизонтальне масштабування, а об'єму наданих ресурсів в межах одного екземпляру – вертикальне. Масштабування може відбуватися вгору при зростанні навантаженні, або вниз – при недостатній утилізації виділених ресурсів. Гібридний тип масштабування надає можливість масштабувати застосунки як вертикально, так і горизонтально залежно від конфігурації, що надає переваги обох підходів.



Рисунок 1.11 – Класифікація масштабування за типом розподілення ОР

За типом ініціації масштабування може бути реактивним, проактивним, за розкладом та гібридним [30]. Основна проблема при автоматизації масштабування є отримання відповідей на питання «коли масштабувати» і «яку кількість ресурсів необхідно надати» [31].



Рисунок 1.12 – Класифікація масштабування за типом ініціації

Реактивне масштабування покладається на поточні метрики рівня QoS та утилізації ресурсів. Падіння рівня QoS, наявність помилок нестачі ресурсів, тротлінг або занадто високий рівень утилізації є сигналами до збільшення кількості екземплярів застосунку. При низькій утилізації ОР автоматично

здійснюється масштабування відповідно до поточних потреб. Цей тип масштабування реагує на зміни в стані застосунку або навантаженні, забезпечуючи оперативне усунення виникаючих проблем, але тільки при наявності даних моніторингу. Реактивне масштабування є простим у впровадженні, оскільки не вимагає алгоритмів прогнозування для своєї роботи. Головний недолік реактивного масштабування це затримка між появою сигналу і безпосередньо розгортанням нових екземплярів, що може призводити до тимчасових порушень у роботі застосунку. Реактивне масштабування найкраще підходить для застосунків із плавними та статичними навантаженнями, але може бути менш ефективним у випадках різких змін навантаження. Реактивне масштабування є важливою частиною СУ ІТ-інфраструктурою, забезпечуючи базову автоматизацію управління ресурсами [32].

Таблиця 1.1 – Порівняння типів масштабування за типом ініціалізації

Тип ініціації	Час масштабування	Об'єм наданих ресурсів
Реактивне	Одразу при погіршенні рівня QoS, недостатній утилізації, перевищення запитів	Відповідно до поточного рівня потреб
Проактивне	Визначається на основі прогнозування навантаження на застосунок; в прогнозовані моменти зміни навантаження	Визначається в залежності від прогнозу навантаження на застосунок та прогнозованих потреб
За розкладом	За встановленими правилами	За встановленими правилами

Проактивне масштабування базується на прогнозуванні майбутніх навантажень і здійснюється до фактичного збільшення або зменшення об'ємів необхідних для роботи застосунків ресурсів [33]. Це дозволяє підготуватися до змін заздалегідь, що значно знижує ризик виникнення тимчасових порушень у роботі застосунків через затримки розгортання. Водночас

проактивне масштабування вимагає наявності точних прогнозів майбутніх об'ємів ресурсів і припускається, що шаблон навантаження не є виключно випадковим. В таблиці 1.1 проводиться порівняння типів масштабування за часом.

Автоматизація управління ресурсами може бути описана за допомогою циклу моніторинг-аналіз-планування-застосування [34], як це зображено на рис. 1.13.

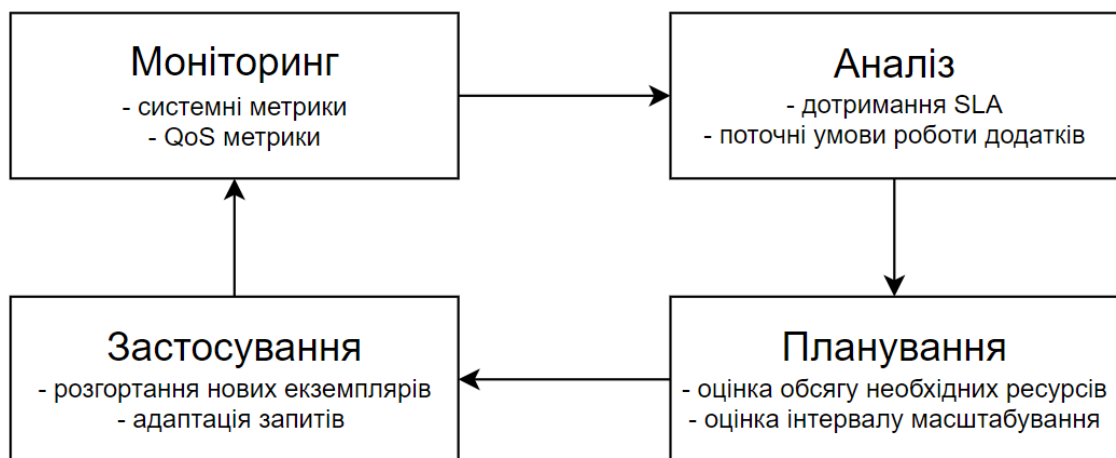


Рисунок 1.13 – Цикл автоматизованого управління ресурсами

Моніторинг включає збір системних та високорівневих метрик, їх агрегацію та обробку для подальшого використання. На етапі аналізу проводиться аналіз поточного стану застосунку на основі оброблення зібраних метрик, зокрема – дотримання SLA, рівня утилізації ОР, існуючих трендів або сезонності в метриках. На основі аналізу здійснюється планування, під час якого визначається необхідний об'єм ресурсів для подальшої роботи та час застосування нової конфігурації ОР. Після впровадження змін в конфігурації проводиться моніторинг стану ІС для оцінки ефективності внесених змін. При проактивному масштабуванні етапи аналізу і планування включають в себе прогнозування майбутніх конфігурацій ОР та визначення розкладу масштабування.

Серед інструментів управління ОР можна виділити встановлення лімітів на ОР додатково до запитів. Якщо запити гарантують наявність ОР на вузлі, то ліміти дозволяють використовувати вільні об'єми поза запитами, що може

додатково підвищувати стабільність роботи застосунків. Прикладом такої реалізації є механізм `requests i limits` в Kubernetes. Пріоритезація також є важливим інструментом управління ОР в ІС, оскільки дозволяє визначити які застосунки або сервіси можуть бути згорнуті або видалені в першу чергу при недостатньому об'єму ресурсів на вузлі або в кластері в цілому.

### 1.3.2 Особливості горизонтального масштабування

Горизонтальне масштабування є основним методом управління ресурсами в хмарних обчисленнях, при якому кількість вузлів кластера або екземплярів застосунку адаптуються до поточного навантаження задля підтримки необхідного рівня QoS та утилізації зарезервованих ОР. Горизонтальне масштабування не обмежено фізичними ресурсами одного сервера на відміну від вертикального. Але не всі застосунки можуть підтримувати цей метод масштабування, наприклад, бази даних, які прив'язані до диску на фізичній машині. Тому для використання даного методу основною вимогою є еластичність компонента, що масштабується, та можливість розподілення навантаження між його екземплярами [35].

Горизонтальне масштабування надає гнучкість та відмовостійкість ІС, оскільки дозволяє розподіляти навантаження між багатьма серверами або екземплярами. У разі відмови частини серверів або екземплярів навантаження може бути перерозподілене на інші через повне дублювання функціоналу, що забезпечує високий рівень доступності послуг для користувачів. Крім того, такі компоненти можуть бути швидко заміщені без зміни конфігурації ІС.

Незважаючи на чисельні переваги, горизонтальне масштабування збільшує складність управління ІС, оскільки вимагає координації множини вузлів або екземплярів та вимагає комплексних рішень для балансування навантаження між ними. Крім того, даний тип масштабування може призвести до збільшення вартості обслуговування ІС через необхідність підтримки більшої кількості апаратних засобів.

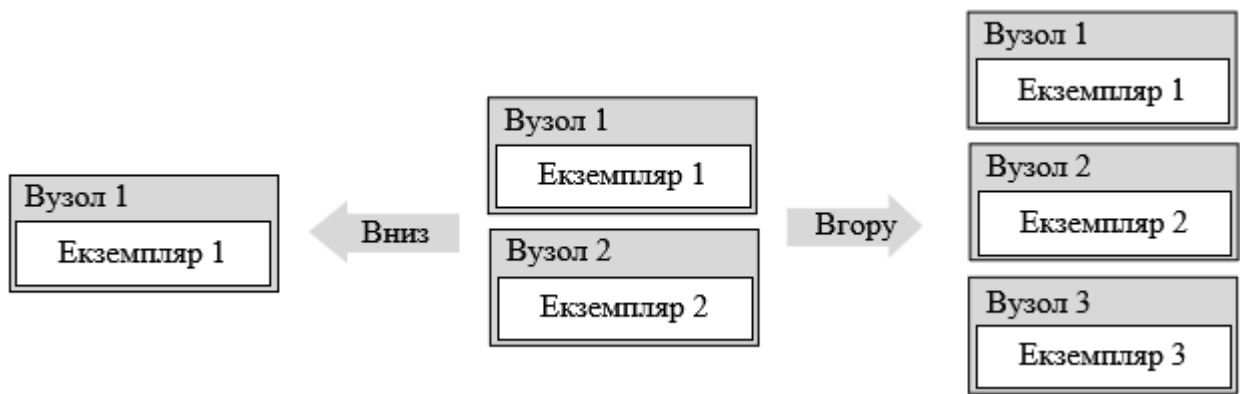


Рисунок 1.14 – Здійснення горизонтального масштабування

Визначено, що  $X_{ly}$  є запитом на  $y$ -й обчислювальний ресурс для  $l$ -го застосунку, де  $y = \overline{1, h}$ , де  $h$  – кількість типів ресурсів для управління, а  $l = \overline{1, m}$ , де  $m$  – кількість застосунків у кластері.  $\hat{X}_{lyj}$  є фактичним споживанням  $y$ -го ресурсу для  $j$ -го екземпляру  $l$ -го застосунку, де  $j = \overline{1, k}$ ,  $k$  – кількість екземплярів  $l$ -го застосунку. Для зручності відображення залежності від часу надалі використовуються функції часу  $k_{ly}(t)$  та  $\hat{X}_{lyj}(t)$ , оскільки вони є динамічними, а  $X_{ly}$  є константним при горизонтальному масштабуванні. Визначимо функцію загального навантаження  $W_{ly}(t)$  залежно від часу на застосунок для кожного з ресурсів:

$$W_{ly}(t) = \sum \hat{X}_{lyj}(t), \quad l = \overline{1, m}, y = \overline{1, h}, j = \overline{1, k}. \quad (1.3)$$

У будь-який момент часу  $W_{ly}(t)$  дорівнює сумі використання ресурсів всіма екземплярами застосунку. В такому випадку кількість екземплярів застосунку в залежності від ресурсу визначається наступною рівністю:

$$k_{ly}(t) = \left\lceil \frac{W_{ly}(t)}{X_{ly}} \right\rceil, \quad l = \overline{1, m}, y = \overline{1, h}, \quad (1.4)$$

де  $k_{ly}(t)$  є граничною кількістю екземплярів застосунку для  $y$ -го типу ресурсу  $l$ -го застосунку. Таким чином при горизонтальному масштабуванні необхідно враховувати потреби по кожному з наявних ресурсів:

$$k_l(t) = \max(k_{ly}(t)), \quad l = \overline{1, m}, y = \overline{1, h}, \quad (1.5)$$



де  $k_{ly}(t)$  – граничною кількість екземплярів  $l$ -го застосунку для забезпечення необхідного об'єму всіх ОР. Додатково необхідно врахувати, що функція навантаження  $W_{ly}(t)$  може виходити за рамки встановлених об'ємів, що призводить до падіння рівня QoS або повної відмови надання сервісу, тому для обробки таких ситуацій необхідно надавати резервний об'єм  $\varepsilon_y$  кожного  $y$ -го типу ресурсу:

$$k_l(t) = \max(k_{ly}(t) \cdot \varepsilon_y), l = \overline{1, m}, y = \overline{1, h}. \quad (1.6)$$

Тоді критерій для масштабування вгору та вниз можна визначити як різницю кількості екземплярів на наступному та поточному кроках

$$\Delta k_l = k_l(t+1) - k_l(t), l = \overline{1, m}, \quad (1.7)$$

де  $\Delta k_l$ , у разі, якщо його значення не дорівнює 0, є індикатором необхідності зміни конфігурації застосунку. При  $\Delta k_l > 0$  відбувається масштабування вгору, а при  $\Delta k_l < 0$  – вниз.

Необхідно зазначити, що навантаження на екземпляри застосунку може розподілятися нерівномірно через наступні причини:

- некоректна робота балансувальника, що призводить до нерівномірного розподілення запитів між екземплярами;
- некоректно обраний алгоритм роботи балансувальника, який не враховує особливості роботи застосунку;
- різна конфігурація вузлів, яка може призвести до значної різниці в продуктивності при обробці запитів, зокрема підтримка більш ефективних інструкцій процесора або стандартів пам'яті;
- використання різних алгоритмів обробки запитів, коли схожі задачі можуть значно відрізнятися в часі обробки.

Через вказані вище причини фактичне використання ОР  $\hat{X}_{lyj}(t)$  екземплярами одного і того самого застосунку може відрізнятися. Для нівелювання некоректного балансування можна ввести коефіцієнт відношення максимального і середнього навантаження:

$$k_l(t) = \max(k_{ly}(t) \cdot \varepsilon_y \cdot \frac{\max(U_{lyj}(t))}{\text{avg}(U_{lyj}(t))}), l = \overline{1, m}, y = \overline{1, h}, j = \overline{1, k}. \quad (1.8)$$

Наведена модель при роботі враховує потреби по всім доступним типам ОР. Проте деякі типи є другорядними і можуть ігноруватися при обчисленні кількості екземплярів. Також, масштабування може відбуватися тільки по одному з ОР, наприклад, `targetCPUUtilizationPercentage` в НРА [36], де обчислення кількості екземплярів відбувається лише за середнім значенням утилізації процесорного часу. Такий підхід може бути корисний, наприклад, у випадку CPU-інтенсивних задач, коли саме процесорний час є обмежуючим ресурсом [37]. Критичність кожного типу ресурсу відома розробникам застосунку та визначається на основі потреб застосунку та доступних ресурсів у кластерів.

Важливо враховувати специфіку управління кожним окремим типом ресурсу. Недостатній об'єм одного з ресурсів може призвести до повної відмови від обслуговування. Наприклад, недостатній об'єм RAM при роботі екземпляру призводить до помилок відділення пам'яті. З іншого боку, при недостатньому об'ємі процесорного часу операційна система буде штучно сповільнювати роботу програми [37].

Розгортання нового екземпляру вимагає здійснення наступних операцій:

- пошук вузла з необхідним об'ємом ОР та резервування ОР відповідно до заданої конфігурації;
- завантаження образів, залежностей та коду застосунку для подальшої ініціалізації контейнера;
- початкова ініціалізація програми – встановлення мережесх'єднань, конфігурація інтерфейсів, початкове кешування даних необхідних для роботи застосунку.

Ці операції можуть займати значний час перед початком безпосередньої роботи екземпляру, що призводить до затримок масштабування та має негативний вплив на рівень QoS і дотримання SLA.

Модель горизонтального масштабування з врахуванням затримок має наступний вигляд:

$$k_l(t) = \max(k_{ly}(t + \eta_{\uparrow})), l = \overline{1, m}, y = \overline{1, h}, \quad (1.9)$$

де  $k_{ly}(t)$  – кількістю екземплярів  $l$ -го застосунку для забезпечення необхідного об'єму всіх ОР з врахуванням майбутніх потреб,  $\eta_{\uparrow}$  – є час затримки масштабування вгору, що визначається як:

$$\eta_{\uparrow} = \eta_{\text{вузол}} + \eta_{\text{образ}} + \eta_{\text{ініціалізація}}, \quad (1.10)$$

де  $\eta_{\text{вузол}}$  – час пошуку вузла, що може включати в себе додавання нового вузла в кластер в разі необхідності,  $\eta_{\text{образ}}$  – час доставки залежностей і коду застосунку з реєстрів на вузол,  $\eta_{\text{ініціалізація}}$  – час на початкову ініціалізацію застосунку.

При масштабуванні вниз дана затримка розгортання екземплярів не враховується, оскільки видалення навантаження відбувається після фактичного зменшення навантаження:

$$k_l(t) = \begin{cases} \max(k_{ly}(t + \eta_{\uparrow})), k_{ly}(t + \eta_{\uparrow}) > k_{ly}(t) \\ \max(k_{ly}(t)), k_{ly}(t + \eta_{\uparrow}) \leq k_{ly}(t) \end{cases}, l = \overline{1, m}, y = \overline{1, h}. \quad (1.11)$$

Реалізація даної моделі можлива лише при проактивному масштабуванні, оскільки включає в себе прогнозування майбутніх навантажень на екземпляри застосунку.

### 1.3.3 Особливості вертикального масштабування

При вертикальному масштабування кількість екземплярів застосунку не змінюється. Адаптація загального об'єму ОР застосунку до динамічних навантажень відбувається внаслідок конфігурації запитів на ОР для кожного екземпляру. Таке масштабування використовується, коли застосунок не може бути масштабований горизонтально, зокрема, бази даних або застосунки з вбудованим сховищем [32].

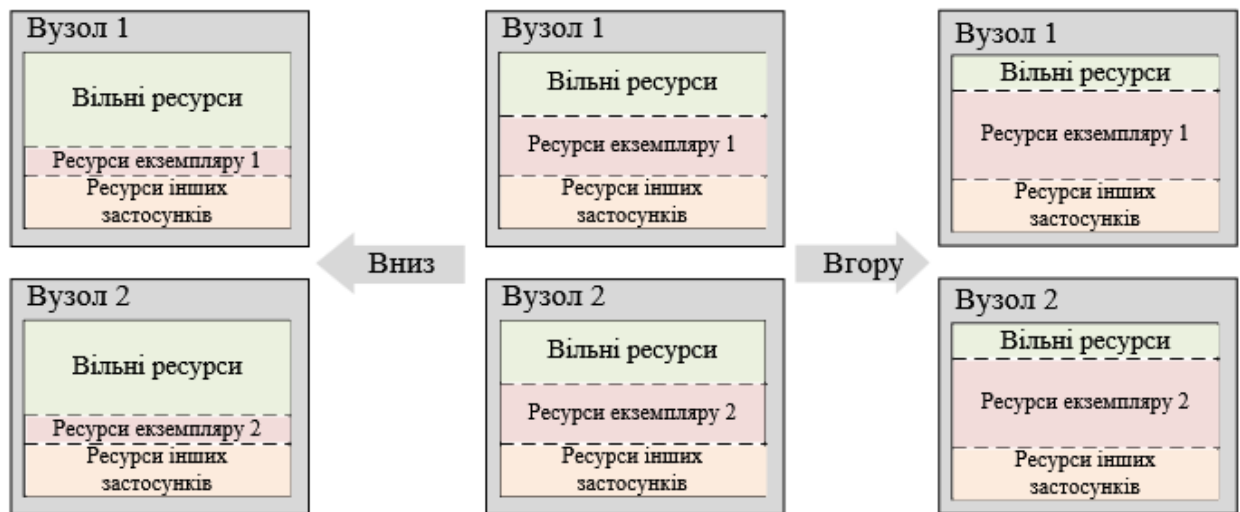


Рисунок 1.15 – Здійснення вертикального масштабування

На відміну від горизонтального масштабування, вертикальне обмежується об'ємом ОР фізичної машини, на якому розміщується екземпляр застосунку. Тому зміна запитів на ОР може потребувати перезапуску екземпляру застосунку через обмеженість компонентів управління ресурсами ОС, наприклад, `cgroups` у Linux, або при необхідності збільшити розмір фізичної машини.

На відміну від горизонтального масштабування, функція кількості екземплярів  $k_l(t)$  для  $l$ -го застосунку має постійне значення. Об'єм зарезервованого  $y$ -го типу ресурсу є функцією  $X_{ly}(t)$ , яка визначається наступним чином:

$$X_{ly}(t) = \left\lceil \frac{W_{ly}(t)}{k_{ly}} \right\rceil, l = \overline{1, m}, y = \overline{1, h}, \quad (1.12)$$

де  $h$  – кількість типів ресурсів для управління,  $m$  – кількість застосунків у кластері,  $W_{ly}(t)$  – функція навантаження для  $y$ -го типу ресурсу  $l$ -го застосунку.

При наявності кількох екземплярів застосунку для яких застосовується вертикальне масштабування, необхідно враховувати максимальне використання кожного типу ОР, оскільки при балансуванні запитів між екземплярами не завжди вдається рівномірно розподілити запити:

$$X_{ly}(t) = \max(X_{lyj}(t)), l = \overline{1, m}, y = \overline{1, h}, j = \overline{1, k}, \quad (1.13)$$

де  $h$  – кількість типів ресурсів для управління,  $m$  – кількість застосунків у кластері, а  $j$  – кількість екземплярів  $l$ -го застосунку.

Проактивна модель вертикального масштабування є аналогічною моделі (1.11) горизонтально масштабування та визначається наступним чином:

$$X_{ly}(t) = \begin{cases} \max(X_{ly}(t + \eta_{\uparrow})), X_{ly}(t + \eta_{\uparrow}) > X_{ly}(t) \\ \max(X_{ly}(t)), X_{ly}(t + \eta_{\uparrow}) \leq X_{ly}(t) \end{cases}, l = \overline{1, m}, y = \overline{1, h}, \quad (1.14)$$

де  $h$  – кількість типів ресурсів для управління,  $m$  – кількість застосунків у кластері,  $j$  – кількість екземплярів  $l$ -го застосунку,  $\eta_{\uparrow}$  – час затримки масштабування вгору.

Моделі вертикального і горизонтального масштабування в загальному вигляді є схожими. Тому з'являється можливість побудувати універсальне рішення для реалізації обох методів масштабування.

## **1.4 Аналіз задач управління обчислювальними ресурсами при управлінні якістю послуг**

### **1.4.1 Обґрунтування необхідності дослідження та розробки методів масштабування**

При аналізі задач управління ОР спочатку необхідно сформулювати контекст, в якому розглядається управління ОР. В незалежності від призначення ІС – бізнес, наукове чи соціальне призначення – її власники зацікавлені в тому, щоб надавати належний рівень QoS при найменших фінансових витратах. Тому при вирішенні задач, що розглядаються в дисертації, враховується критерій забезпечення стабільного рівня якості сервісів при мінімізації об'єму ОР, що використовуються застосунками.

Однією із задач управління ОР у кластері є автоматизація управління розподіленням ОР. Основним інструментом управління ОР, який розглядається в роботі, є масштабування застосунків. Недостатнє виділення ОР або кількості екземплярів застосунку призводить до погіршення показників якості сервісів і в результаті до порушення встановлених вимог

SLA щодо якості IT-послуг, що несе за собою фінансові та репутаційні втрати. В той же час, надмірне виділення ресурсів призводить до неефективного використання ОР і, як наслідок, фінансових ресурсів, які можна було б використати для вирішення інших обчислювальних задач. Підтримка балансу між вказаними двома аспектами є задачею динамічного розподілу ресурсів. Сучасні ІС можуть включати тисячі різних застосунків, які мають свої особливості та потреби в ОР, що значно ускладнює вирішення задачі управління ОР.

Однією із задач управління ОР є прогнозування робочих навантажень у СККЗ, оскільки наявність даних про майбутні навантаження надає можливість значно ефективніше планувати розподілення ОР для застосунків та загальний об'єм кластера для обробки потенційного навантаження [38]. Наявність високоточних прогнозів є важливою умовою для забезпечення необхідного рівня QoS та ефективної утилізації наявних ОР. Робочі навантаження мають одну або декілька сезонностей – шаблонів навантажень, що повторюються із постійною періодичністю та можуть накладатися – різної тривалості, тенденції зростання або зменшення та інші особливості [39]. На основі аналізу часових рядів можна прогнозувати кількість запитів користувачів і, відповідно, майбутні потреби застосунків в ОР. Аналіз існуючих алгоритмів прогнозування та їх адаптація до управління обсягами ОР в залежності від робочого навантаження у кластері є однією з підзадач даної дисертації. Для цього необхідно визначити підходи до валідації та оцінки прогнозів в потребах ОР.

Також необхідно розглянути задачу ідентифікації аномалій робочих навантажень у кластері. Нетипове підвищення або зниження робочих навантажень може статися через низку причин, зокрема, через мережеві збої, відмови від обслуговування компонентів ІС, зміни в архітектурі, сезонні коливання або свята. Такі події здебільшого неможливо передбачити, а їх виникнення призводить до значного погіршення значень показників QoS у період реагування СУІ або до значної збитковості резервування ОР. СУІ має

швидко реагувати на аномальні події та відповідно і адаптувати обсяги ОР до поточних потреб. Вирішення даної задачі є важливим при інтеграції реактивних і проактивних підходів до масштабування. Однією із задач даної дисертації є пошук рішень, які дозволяють вчасно виявити аномалії в роботі застосунків та виконувати відповідні заходи для стабілізації процесу надання ІТ-послуг ІС.

Розробка проактивних методів масштабування застосунків є актуальною проблемою в екосистемі Kubernetes, оскільки стандартні інструменти не надають такої можливості [40], а існуючі рішення з використанням реактивних методів базуються на евристичних, тому неточних алгоритмах. Проактивні методи масштабування є більш ефективними у порівнянні з реактивним в контексті управління ОР в ІС з декількох причин [33]. По-перше, проактивні методи управління дозволяють завчасно масштабувати застосунки під час пікових навантажень завдяки, наприклад, аналізу історичних даних. Це дозволяє значно зменшити ризик погіршення показників QoS за умов суттєвої динаміки запитів користувачів. По-друге, це дозволяє значно зменшити надмірне резервування ОР завдяки прогнозуванню необхідного об'єму ОР та завчасному плануванню масштабування вниз. Реактивне масштабування вверх здійснюється за фактом появи надмірного навантаження. Втрати часу на розгортання та початкову ініціалізацію застосунку може призвести до тимчасового критичного падіння показників якості ІТ-послуг, навіть до повної відмови. Проактивний підхід, на відміну від реактивного не є самодостатнім і має декілька передумов для його застосування – наявність історичних даних або достовірних прогнозів. У таких випадках коректна робота застосунків з ефективним використанням ОР може бути забезпечена шляхом інтеграції проактивного і реактивного методів масштабування. Важливість цієї задачі обумовила необхідність її вирішення в дисертаційній роботі.

При наявності високоточних прогнозів проактивні методи дозволяють завчасно керувати об'ємом ОР або кількістю екземплярів, що дозволяє оперативно реагувати на динамічні зміни навантажень. Проте низка факторів,

наприклад, аномальні навантаження, втрата або недоступність історичних даних, зміни конфігурації IT-інфраструктури, призводять до суттєвого погіршення точності прогнозів майбутніх навантажень. В таких випадках реактивні методи дозволяють стабілізувати систему доти, поки знов не з'являться точні дані про майбутні робочі навантаження. Гібридний метод, що поєднує проактивний і реактивний компоненти масштабування, завдяки комбінування, може нівелювати описані недоліки обох методів. Основною задачею для реалізації комбінованого підходу IT управління ресурсами є розробка методу визначення необхідності зміни режиму управління в СККЗ.

Можливості вертикального масштабування обмежені ОР одного вузла. Горизонтальне масштабування позбавлено цього недоліка. Проте, вертикальний тип масштабування може бути корисним для підтримання необхідної пропорції ОР, що дозволить зменшити збиткове резервування ресурсів. Крім того, вертикальний метод є необхідним для запобігання відмовах від обслуговування через нестачу критичних ресурсів, зокрема, пам'яті та ООМ помилок. Дана задача полягає в координуванні горизонтального та вертикального масштабування для збільшення ефективності використання ОР і оперативного усунення помилок нестачі ОР.

Ресурси кластера мають обмежену кількість вузлів і ОР, що необхідно враховувати при плануванні використання ОР. Хмарні провайдери надають можливість додавати нові вузли в кластер в ході роботи у разі потреби. Використовуючи прогнозування робочих навантажень, можливо масштабувати не тільки застосунки, а також кластер. Необхідно мати можливість масштабуватися, коли поточних ресурсів кластера не вистачає для забезпечення всіх необхідних робочих навантажень, тому дана задача полягає в автоматизації додавання вузлів в кластер із застосуванням проактивних методів для зменшення затримок розгортання нових екземплярів застосунків і підтримання необхідних QoS.



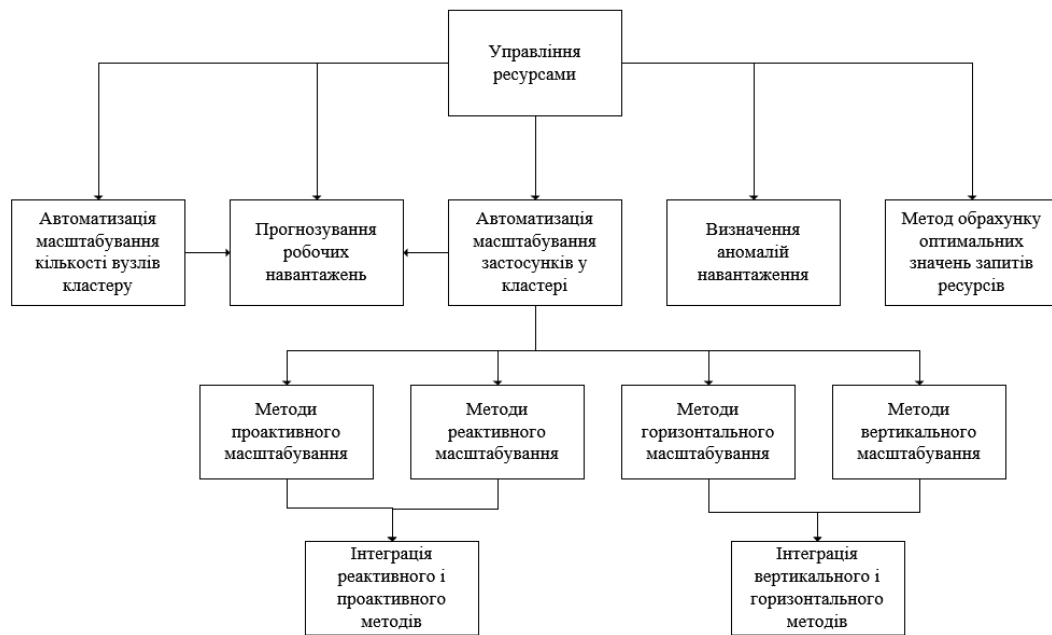


Рисунок 1.16 – Діаграма задач управління ОР у кластері

На рис. 1.16 зображено діаграму послідовності вирішення задач. Зокрема, для розробки проактивних методів необхідно спочатку мати моделі прогнозів навантаження, а також їхньої валідації. Інші задачі є комбінацією двох методів для покращення сумарних властивостей, зокрема інтеграція проактивних і реактивних методів.

#### 1.4.2 Визначення функціоналу рішень для управління ресурсами

Актуальні рішення для управління ОР в кластері є комплексними системами, які включають компоненти моніторингу стану системи та окремих застосунків, аналізу ефективності утилізації ресурсів, прогнозування майбутніх об'ємів навантажень, планування масштабування та безпосередньо керування розподіленням наявних ресурсів. Основні задачі ІТ та СККЗ є наступними:

- моніторинг використання ОР застосунками, запитів на ОР, наявного вільного об'єму ОР та працездатності компонентів, в тому числі помилок в роботі застосунків, що викликані некоректним розподіленням ОР;
- обробка, агрегація та зберігання даних моніторингу з метою подальшого аналізу при плануванні розподілення ОР між застосунками кластера;

- відстежування рівня QoS та його контроль у разі необхідності шляхом адаптації вертикальної та горизонтальної конфігурації компонентів цільової системи;
- прогнозування майбутніх навантажень та потреб у ОР з метою подальшого планування та проактивного управління ОР, прогнозування потенційних критичних станів ІС;
- планування розподілення наявних ОР та розширення кластера у разі необхідності;
- управління розподіленням ОР шляхом вертикального та горизонтального масштабування і пріоритезацією компонентів;
- реагування та оперативне усунення помилок в роботі компонентів ІС викликаних некоректним розподіленням ОР або аномальним об'ємом навантаження;
- сигналізування про наявні або потенційні проблеми при управлінні ОР, які не можуть бути усунені в автоматичному режимі.

Крім функціоналу ІТ та системи управління ОР визначимо вимоги до ІТ та системи управління ОР, які обумовлюють її здатність працювати в умовах сучасних парадигм надання ІТ-послуг:

- здатність працювати незалежно від архітектури та виробника процесора, пам'яті та інших типів ресурсів, наприклад, підтримка як X64 та ARM, що дозволить працювати з гібридними кластерами, які включають різні технології, що покращує здатність до масштабування кластера в майбутньому;
- розподіленість та дублювання компонентів системи управління ОР для підвищення надійності, оскільки некоректна робота системи може призвести до некоректного планування, відмови від обслуговування або погіршення рівня QoS;
- висока швидкість реагування на погіршення рівня QoS та помилки, що виникають;
- підтримка різних типів ОР, наприклад, окрім процесорного часу та пам'яті, управління може здійснюватися для пропускної здатності мережі,

наявного дискового простору, спеціальних пристроїв, наприклад, графічних процесорів;

- масштабованість компонентів системи управління ОР з метою підтримки значної кількості вузлів кластера і відповідно значної кількості застосунків розміщених на даних вузлах.

На основі визначених функціональних та нефункціональних вимог можемо визначати структуру системи управління ОР.

## **Висновки до розділу 1**

На основі проведеного аналізу поточного стану і проблем управління ОР, а також особливостей управління ІТ-інфраструктурою в цілому визначені актуальні проблеми управління ОР та функціонал ІТ та СУ, що їх вирішують.

Формалізовано загальну модель кластера, в якій виконуються процеси розміщення контейнерів на вузлах, оновлення конфігурації ресурсів, моніторингу та балансування навантаження. На основі моделі визначені обмеження та інструменти, які необхідно використовувати при розробці методів та інформаційної технології управління в подальшому.

Проведено порівняльний аналіз віртуалізації та контейнеризації. Визначені переваги та недоліки обох методів розгортання програмних застосунків на фізичних або віртуальних машинах. Аналіз дозволив обґрунтувати використання саме технології контейнеризації для швидкого і гнучкого управління розгортанням застосунків.

Розглянуто існуючі засоби для управління ОР в кластерах. Здійснено аналіз горизонтального і вертикального масштабування, їх переваг, недоліків та можливих сценаріїв використання. Описано аналітичну модель масштабування.

## 2 РОЗРОБКА ПРОАКТИВНИХ МЕТОДІВ МАСШТАБУВАННЯ

Проактивне масштабування – метод управління ОР в ІС з прогнозуванням майбутнього навантаження на застосунок та попередньої підготовки до збільшення ресурсів, що надаються. Методи проактивного масштабування необхідні у контексті дотримання вимог SLA, наприклад, задля гарантування доступності сервісів на рівні 99.99% часу календарного року.

Одним із можливих варіантів забезпечення високих показників доступності незалежно від навантаження є резервування необхідного об'єму ОР протягом всього часу роботи. Проте, в такому випадку необхідно враховувати високі фінансові витрати. Реактивне масштабування частково вирішує дану проблему шляхом адаптації, але в той же час може призводити до тимчасових падінь рівня QoS під час пікових навантажень. Головним недоліком реактивного підходу є додатковий час на масштабування вгору через такі обставини [32]:

- у разі відсутності доступного для розгортання вузла в кластері з необхідним об'ємом ОР здійснюється запит на ВМ відповідного рівня до хмарного провайдера, відбувається додавання нового вузла та його початкова ініціалізація;
- інсталяція на вузлі початкових даних для роботи, зокрема, образів застосунків або необхідних бібліотек та коду;
- початкова ініціалізація застосунку, мережових інтерфейсів, отримання кешованих даних.

Також при використанні реактивних методів необхідно враховувати час реакції, оскільки збір та обробка системних чи інших метрик займає деякий час. Проактивні методи дозволяють уникнути цих проблем реактивних методів, оскільки процес додавання об'єми ресурсів відбувається з врахуванням часу розгортання, що важливо для підтримання QoS на відповідному рівні [33].

## 2.1 Аналіз особливостей проактивного масштабування

Основною перевагою проактивного масштабування є мінімізація надмірного резервування ОР при одночасному підтриманні необхідного рівня QoS. ІС включає значну кількість застосунків, кожен з яких має свої унікальні особливості роботи та вимоги до ОР. Проактивне масштабування, з одного боку, дозволяє значно покращити показники QoS, а з іншого – підвищити ефективність використання ОР [41].

Основною передумовою застосування проактивного масштабування є наявність методу прогнозування, що здатний враховувати комплексні сезонності, тенденції та є стійким до нетипових ситуацій в історичних даних [38]. Відповідно наявність історичних даних є необхідною умовою застосування даного методу. Для роботи з довгими сезонностями необхідно мати значний об'єм даних, що вимагає застосування спеціалізованих рішень для обробки і зберігання метрик утилізації ОР. Також необхідно мати дані про час розгортання екземплярів застосунку, додавання нових вузлів та часові витрати на ініціалізацію застосунку для розрахунку затримок при масштабуванні застосунку.

До недоліків проактивних методів можна віднести необхідність розгортання інфраструктури для моніторингу, зберігання та обробки значних об'ємів історичних даних. Крім того, під час нетипових навантажень методи прогнозування можуть бути неефективними [39]. Тому проактивний метод недоцільно застосовувати для застосунків, в яких навантаження не є сезонним або не містить шаблонів. Також, деякі параметри даних методів налаштовуються вручну, зокрема, сезонності не завжди можуть бути коректно ідентифіковані в існуючих даних.

Життєвий цикл проактивних методів включає етапи, що зображено на рис. 2.1. Робота починається зі збору даних моніторингу поточного стану ІС – системних метрик використання ресурсів або високорівневих метрик, їх попередня агрегація та зберігання

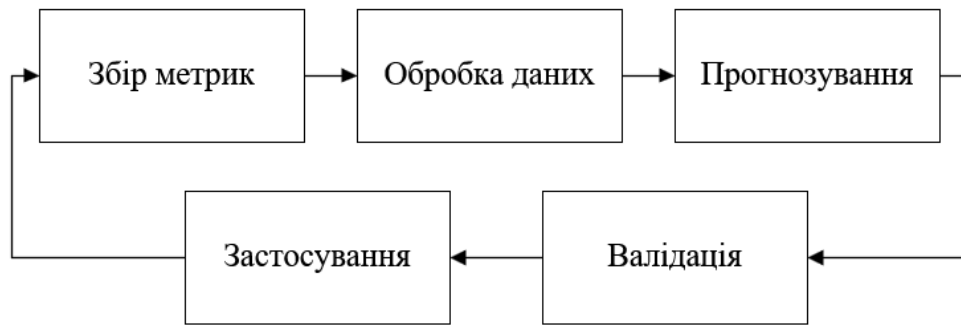


Рисунок 2.1 – Життєвий цикл роботи проактивного масштабування

На наступному етапі здійснюється обробка часових рядів для подальшого використання в методах прогнозуванні. Після цього відбувається прогноз навантаження на основі отриманих даних. Даний етап включає ідентифікацію аномалій, шаблонів, сезонностей, трендів, підбір параметрів моделі та оцінку її точності. На етапі валідації приймається рішення про доцільність використання отриманих прогнозів. На цьому етапі приймається рішення про використання альтернативних методів управління ОР. У разі успішної валідації прогнозів відбувається застосування нових конфігурацій. Цей процес є ітеративним та заключається в постійному моніторингу поточного стану ІС, відповідності QoS встановленим SLA та адаптації конфігурації ІС до поточних умов.

## 2.2 Аналіз існуючих рішень для проактивного масштабування

Один з методів проактивного масштабування будується на основі знаходження найбільш подібного шаблону навантаження в минулому і його екстраполяції на поточний стан. Наприклад, в роботі [42], автори пропонують рішення, в якому історичний часовий ряд аналізується на предмет закономірностей і на їх основі відбувається пошук найбільш подібного шаблону навантаження до поточного. Виявлені закономірності можуть відрізнитися за масштабом, але за співвідношенням між елементами ідентифікованого шаблону та поточного шаблону мають бути подібними. Отримані шаблони інтерполуються за допомогою зваженої інтерполяції. Найбільш подібні шаблони мають найбільшу вагу в результуючому часовому

ряді. По суті, даний метод зводиться до пошуку найбільш подібної ситуації навантаження у минулому та його адаптації до поточного навантаження. Перевагою методу є можливість прогнозування нетривіальних часових рядів, в який відсутня будь-яка сезонність, але в випадкові моменти часу можуть відбуватися типові для даного оточення відрізки навантаження. Основним недоліком даного методу є низька точність прогнозування у порівнянні з іншими методами прогнозування та некоректність роботи у випадках, коли шаблон навантаження не зустрічається в історичних даних [43].

В роботі [44] розглядається проактивне масштабування на основі ARMA/ARIMA. Автори пропонують метод, в якому є кілька рівнів впевненості, які обираються в залежності від особливостей застосунку. Також проводиться оцінка точності отриманої моделі та оцінка впливу на показники якості послуг. Варто зазначити, що історичні дані не містять комплексної сезонності, відповідно оцінка відбувається на простому часовому ряді. Також зазначається, що знаходження коефіцієнту порядку авторегресії, що визначає кількість попередніх значень часової серії, які використовуються для прогнозування поточного значення, та коефіцієнту порядку ковзного середнього, для даного методу прогнозування є нетривіальною задачею з точки в контексті часу обчислення та об'єму ОР.

В роботі [45], автори запропонували двокомпонентний дизайн PRESS для проактивного масштабування. Перший компонент застосовується для шаблонів навантажень, які містять приклади навантажень, що повторюються. Для обробки такого типу шаблонів використовується швидкі перетворення Фур'є, а для порівняння схожості поточних і минулих прикладів критерій Пірсона. Якщо критерій не дає необхідного показника схожості, то використовується другий компонент «на основі стану». Для цього компонента використовуються ланцюги Маркова, де всі можливі варіанти рівномірно розподіляються серед заданої кількості корзин. Після цього будується граф можливих станів і матриця вірогідностей. Автори стверджують, що їх рішення легко масштабується і підходить для комплексних систем.

В роботі [46] для управління якістю послуг використовуються нейронні мережі, а саме довготривала короткочасна пам'ять. Мережі довготривалої короткочасної пам'яті – це вид рекурентних нейронних мереж, здатних вивчати довготривалі залежності. Особливістю таких мереж є здатність знаходити комплексні залежності та зберігати інформацію протягом довгих послідовностей або періодів часу. Тому такі мережі добре вирішують задачі прогнозування часових рядів. Крім того, метод доповнений навчанням з підкріпленням для отримання більш точних прогнозів. Недоліками методів на основі нейронних мереж – є необхідність великих об'ємів даних для навчання і неможливість пояснити рішення моделі.

### **2.3 Аналіз методів прогнозування**

Основним компонентом рішення для проактивного масштабування є механізм прогнозування. В даному підрозділі проводиться аналіз методів прогнозування на предмет їх точності в умовах хмарних навантажень, наявності аномальних шаблонів та обмежених історичних даних [38].

Припускається, що шаблон навантаження компонентів ІС має чітку сезонність або тенденцію. Тому перша вимога до проактивних моделей, які розробляються в роботі, це можливість працювати з однією чи кількома сезонностями. Також ці моделі мають бути точними, оскільки чим точніший прогноз, тим точніше розраховується необхідний об'єм виділених ОР.

Зважаючи на існуючий тренд використання в ІС мікросервісів, доцільно масштабувати всі компоненти кластера. Кожен компонент мікросервісної архітектури має унікальні особливості роботи та функціонал, тому шаблон навантаження є індивідуальним для кожного компонента, а таких компонентів може бути велика кількість. Це приводить до висновку, що обробка історичних метрик, а також налаштування параметрів методу прогнозування в ручному режимі для кожного окремого компонента є процесом, який неможливо масштабувати.



Головна перевага та особливість проактивних методів в тому, що з'являється можливість оцінити навантаження в будь-який момент часу у майбутньому. Це означає, що можна адаптувати цільовий компонент до майбутніх навантажень як з точки зору швидкодії, так і з точки зору економії ОР. Проте прогнозування в контексті навантаження не завжди є точними з багатьох причин [39].

Навантаження на компонент залежить від багатьох технічних та нетехнічних факторів – стабільність мережі, доступність центру обробки даних, свята та навіть політична та економічна ситуації. Жоден метод прогнозування не може врахувати всі ці фактори, але її можна зробити адаптивною та стійкою до таких речей. Якщо модель є точною, але навчається занадто довго, то ефективність її застосування також падає. Методи прогнозування мають бути універсальними, надавати точні прогнози, підтримувати комплексні сезонності та тренди.

Метод прогнозування TBATS [47] розроблено для прогнозування складних часових рядів з декількома сезонностями різної довжини. У TBATS до оригінального часового ряду застосовується статистичні трансформації, що полегшують подальше виявлення особливостей часового ряду. Після чого, трансформований часовий ряд розкладається як лінійна комбінація експоненційно згладженого тренду, сезонних компонентів та компонентів ARMA [48]. Сезонний компонент відтворюється з використанням тригонометричних перетворень та рядів Фур'є. Одною з переваг методу є автоматичний пошук довжин сезонностей та інших параметрів. Метод TBATS обрано через його універсальність і здатність адаптуватися до часових рядів різної складності, що означає відсутність необхідності адаптувати параметри методу під навантаження кожного існуючого компонента в системі.

Prophet є бібліотекою на основі машинного навчання для прогнозування часових рядів, яку розроблено у Facebook [49]. Метою розробки було створити простий, прозорий та зрозумілий алгоритм генерації моделей, який би

дозволяв швидко отримувати достовірні прогнози. Алгоритм базується на адитивній регресивній моделі [50], яка має декілька компонентів:

$$y(t) = g(t) + s(t) + h(t) + e(t), \quad (2.1)$$

де  $g(t)$  – компонент трендів,  $s(t)$  – сезонний компонент,  $h(t)$  – аномальні події, а  $e(t)$  – функція помилки.

Крім адитивної регресійної моделі, в Prophet застосовується перетворення рядів Фур'є. Серед переваг моделі – можливість роботи з довільними часовими рядами, можливість працювати ефективно з великими наборами даних та гнучкість у налаштуванні.

NeuralProphet [51] є бібліотекою для прогнозування часових рядів, розроблений на основі бібліотеки PyTorch, яка є розвитком моделі Prophet та надає аналогічний API для використання. Основна відмінність даної бібліотеки від Prophet – використання глибокого навчання замість статичних методів аналізу часових рядів. В основі NeuralProphet лежить авторегресійна нейронна мережа [52], яка поєднує класичні авторегресійні методи прогнозування часових рядів та сучасні підходи на основі машинного навчання. Крім того, NeuralProphet включає можливість автоматичного підбору параметрів компонентів, має можливість працювати з часовими рядами різної сезонності та дозволяє використовувати додаткові змінні для покращення точності прогнозів.

В даній роботі не використовується SARIMA [53], оскільки даний метод не підтримує роботу з кількома сезонностями в одному часовому ряді. Також процес знаходження оптимальних параметрів моделі є нетривіальною задачею.

Обрані методи прогнозування є перспективними та поєднують статистичні та чисельні підходи з сучасними трендами використання машинного та глибокого навчання.

Для оцінки точності методів прогнозування часових рядів доцільно використовувати дві метрики точності – середнє абсолютне відсоткове відхилення (MAPE) та квадратичне середнє відхилення (RMSE). Показник

RMSE дає змогу порівняти відхилення у початкових величинах і допомагає оцінити загальну точність прогнозів:

$$RMSE = \sqrt{\frac{\sum_{t=1}^n (\hat{x}_t - x_t)^2}{n}}, \quad (2.2)$$

де  $x_t$  – реальне значення в момент часу  $t$ ,  $\hat{x}_t$  – прогноз у момент часу  $t$ , а  $n$  – кількість точок даних у наборі даних.

Показник MAPE у відсотках дає змогу порівняти прогнози різних моделей на різних масштабах або даних:

$$MAPE = \frac{100}{n} \sum_{t=1}^n \left| \frac{\hat{x}_t - x_t}{x_t} \right|, \quad (2.3)$$

де  $x_t$  – реальне значення в момент часу  $t$ ,  $\hat{x}_t$  – прогноз у момент часу  $t$ , а  $n$  – кількість точок даних у наборі даних.

Оцінка універсальності методів відбувається на основі оцінки метрик точності без додаткового налаштування параметрів, що необхідно для перевірки здатності може адаптуватися до різних шаблонів навантаження.

Для перевірки точності та універсальності моделей використовуватиметься штучно згенеровано часовий ряд з денною та тижневою сезонністю. Така комбінація сезонностей обрана оскільки вона відображає циклічність поведінки людей у суспільстві, що зумовлена суспільними нормами та звичками. У типовому тижні, особливо у бізнес-оточенні, є чітко виражений ритм, зумовлений робочими та вихідними днями. Денна сезонність, в свою чергу, відображає повторюваність дій людей впродовж доби: робочі години, час на відпочинок, сон і так далі. В цілому, будь-яка інша сезонність може бути обрана, а метою дослідження є перевірка роботи моделей на комплексних сезонностях. Частота значень в згенерованому часовому ряді становить 3600 с і має мінімальний вплив на точність моделей у порівнянні з меншою частотою даних.

У першому експерименті проводиться порівняння обраних моделей на прикладі вище описаного часового ряду з двома періодичностями різної довжини – денної та тижневої. Дані попередньо не оброблялися.

Ціль даного експерименту дослідити можливості прогнозування обраними моделями на складних шаблонах навантаження без спотворень даних, а також дослідити вплив розміру історичних даних під час тренування на точність прогнозів. Моделі тренуються на даних різної довжини, які включають один, два та три тижневих періоди.

Результати першого експерименту, що зображений на рис. 2.2, демонструють точність методів Prophet, NeuralProphet та TBATS на тритижневих даних.

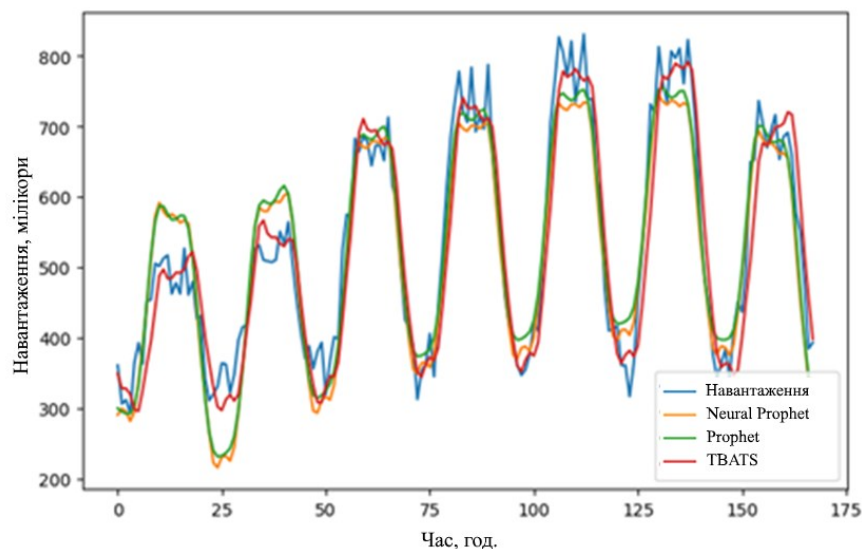


Рисунок 2.2 – Прогнозування денних та тижневих коливань без спотворень даних на основі тритижневих даних

В таблиці 2.1 наводяться показники точності методів, що отримані на основі тритижневих даних. По отриманим графікам і значенням точності можна зробити висновок, що кожна з моделей доволі точно передбачає навантаження в даних умовах. Проте варто зазначити, що TBATS є точнішим на 6% за інші моделі, які показали однаковий результат. Метрики RMSE і MAPE наведені в таблиці 2.1.

Таблиця 2.1 – Прогнозування денних та тижневих коливань без спотворень даних на основі тритижневих даних

Модель	RMSE	MAPE
TBATS	32.464	0.081
Prophet	46.16	0.0856
NeuralProphet	53.705	0.0891

Другий експеримент проводиться на двотижневих даних, результати якого зображені на рис. 2.3. Дані аналогічні першому експерименту. Очікується, що точність моделей не має погіршитися, оскільки два періоди сезонності є граничною довжиною даних для виявлення чітких залежностей в часовому ряді.

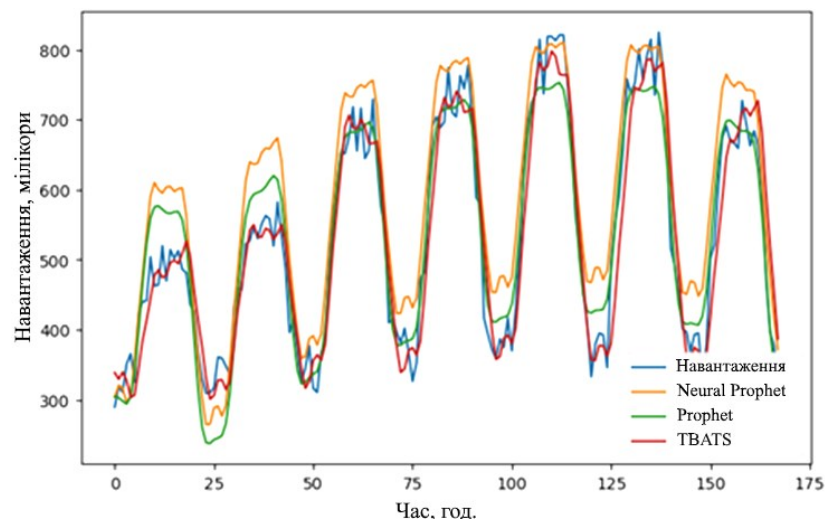


Рисунок 2.3 – Прогнозування денних та тижневих коливань з двома вхідними періодами

Зменшення тривалості даних для тренування майже ніяк не вплинуло на точність у випадку TBATS та Prophet відносно результатів першого експерименту та становить менше 1%. Точність NeuralProphet погіршилась на 44% у порівнянні з першим експериментом, що свідчить про залежність даного методу прогнозування від довжини даних. В таблиці 2.2 наведені результати другого експерименту.

Таблиця 2.2 – Прогнозування денних та тижневих коливань без спотворень даних на основі двотижневих даних

Модель	RMSE	MAPE
TBATS	32.943	0.084
Prophet	51.236	0.089
NeuralProphet	71.378	0.130

Третій експеримент містить аналогічні дані до перших двох, але їх довжина зменшена до одного періоду тижневої сезонності. Очікується, що точність методів має значно зменшитися, оскільки довжина даних є меншою за граничний показник, що становить дві сезонності.

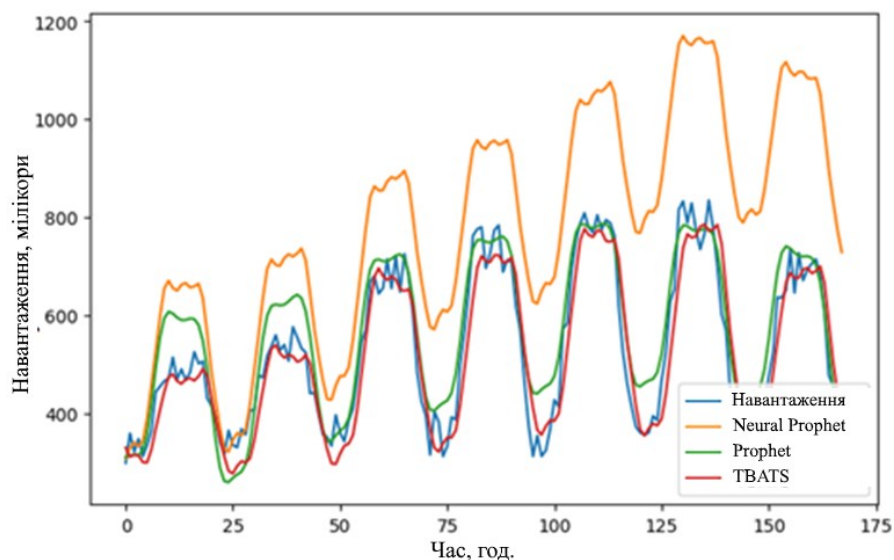


Рисунок 2.4 – Прогнозування денних та тижневих коливань з одним вхідним періодом

В таблиці 2.3 наведені метрики точності RMSE та MAPE. У випадку TBATS падіння точності мінімальне та становить менше 2%. Точність Prophet знизилась на 25% у відносному та 3% у абсолютному вимірі. Точність прогнозування NeuralProphet становить 49% в метриці MAPE, що є критично низьким показником. Прогноз NeuralProphet на рис. 2.3 повністю відповідає низьким метрикам точності.

Таблиця 2.3 – Прогнозування денних та тижневих коливань без спотворень даних на основі однотижневих даних

Модель	RMSE	MAPE
TBATS	35.006	0.085
Prophet	64.427	0.117
NeuralProphet	281.056	0.493

В експерименті, що зображений на рис. 2.5, в історичні дані внесено спотворення. Деяка частина днів мають нетипові збільшені або зменшені значення. Сезонність аналогічна минулим експериментам. В реальних ІС причиною таких спотворень можуть бути свята, відмова мережевого обладнання або проблеми з балансуванням навантаження. В даному експерименті перевіряється стійкість обраних методів до аномалій в історичних даних.

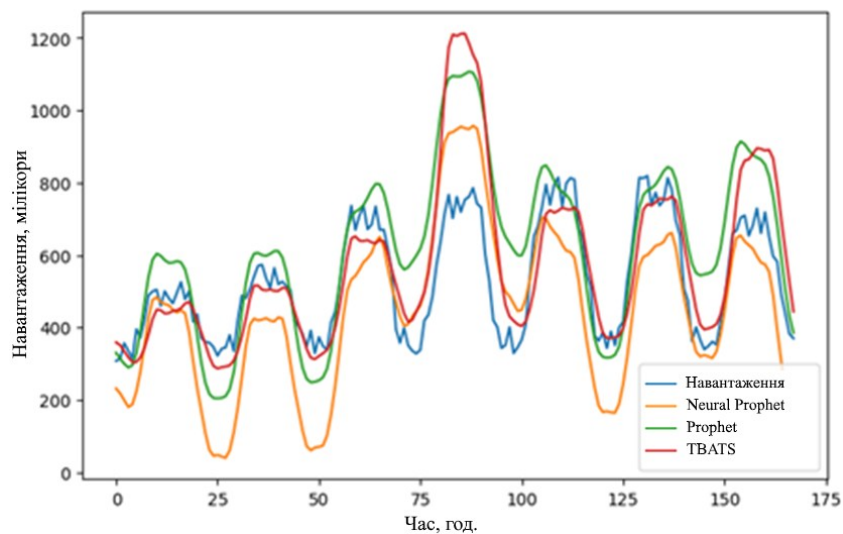


Рисунок 2.5 – Прогнозування денних та тижневих коливань з трьома вхідним періодами та наявністю аномалій у 10% днів

У даному експерименті показники точності прогнозів TBATS у порівнянні з першим експериментом погіршується вдвічі, для Prophet та NeuralProphet зниження точності більше 200%. Загальний шаблон

зберігається, проте наявні значні відхилення. Метрики точності наведені в таблиці 2.4.

Таблиця 2.4 – Прогнозування денних та тижневих коливань з трьома вхідним періодами та наявністю аномалій у 10% днів

Модель	RMSE	MAPE
TBATS	135.949	0.174
Prophet	163.716	0.263
NeuralProphet	161.726	0.297

В наступному експерименті кількість аномальних днів збільшується до 20%, що необхідно для перевірки роботи методів в умовах низької якості даних та відсутності попередньої обробки. Всі інші умови повторюють попередній експеримент. На рис. 2.6 зображені результати прогнозування.

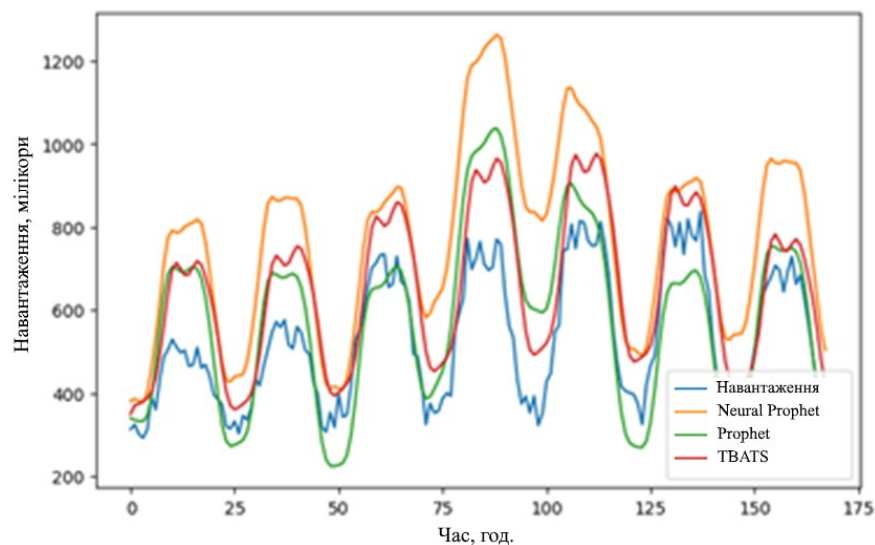


Рисунок 2.6 – Прогнозування денних та тижневих коливань з трьома вхідним періодами та наявністю аномалій у 20% днів

В таблиці 2.5 містяться метрики точності. В таких екстремальних умовах точність прогнозів значно зменшується. Зокрема, точність моделі NeuralProphet в даному випадку є критично низькою. Результати свідчать, що при наявності великої кількості аномалій в історичних даних необхідно проводити попередню обробку даних для отримання високих метрик точності.



Таблиця 2.5 – Прогнозування денних та тижневих коливань з трьома вхідними періодами та наявністю аномалій у 20% днів

Модель	RMSE	MAPE
TBATS	134.313	0.243
Prophet	134.848	0.209
NeuralProphet	274.135	0.476

Результати проведених експериментів показують, що всі три обрані моделі здатні досить точно передбачати складні шаблони навантажень з декількома сезонностями без попередньої обробки даних та із аномаліями. TBATS є більш точним за Prophet та NeuralProphet в усіх проведених експериментах, проте різниця в точності складає не більше 16%. TBATS та Prophet досить точно передбачали навантаження маючи всього лише один тижневий період і погіршення точності складає лише 10%. NeuralProphet потребує додаткового налаштування параметрів або попередньої обробки даних в деяких проведених експериментах. Наявність аномалій значно впливає на точність і попередня обробка історичних даних є необхідною.

Враховуючи отримані результати та поставлені вимоги можна зробити висновок, що всі три обрані моделі можна використовувати для автоматизації масштабування ресурсів в Kubernetes, але необхідно враховувати особливості та передумови їх застосування.

## 2.4 Розробка комбінованого методу прогнозування

Результати проведених експериментів показують, що розглянуті методи прогнозування мають високу точність прогнозів на типових навантаженнях з наявністю сезонностей або трендів. Проте при наявності великою кількості аномалій в історичних даних точність прогнозування зменшувалася з 92% до 76%. Аномалії навантаження можуть мати різне походження, але можна припустити що причини аномальних навантажень є такими, що повторюються. З врахуванням цього припущення з'являється можливість

покращити окремі характеристики розглянутих методів, зокрема, Prophet. Отже, серед переваг розглянутих методів передбачення є можливість роботи з сезонностями та трендами, проте головним недоліком є низька стійкість до нетипових навантажень [43, 54].

Підвищення характеристик стійкості до аномалій має прямий вплив на ефективність проактивних методів, оскільки основний недолік таких методів виникає саме через неможливість спрогнозувати всі можливі події. Підвищення стійкості проактивних методів в перспективі є ключовим аспектом для підтримання необхідного рівня QoS як в цілому, так і у випадку нетипових навантажень.

Вирішення проблеми ідентифікації аномалій у часових рядах є актуальним і розглядається в багатьох роботах [55], зокрема, з використанням статистичних методів [56], нейронних мереж [57] та чисельних методів прогнозування [58], які фокусуються на пошуку аномалій в історичних даних та у реальному часі. Подальшим розвитком цього напрямку є прогнозування аномалій на основі часових рядів, з метою передбачення точного часу та масштабу аномальних подій у майбутньому, зокрема з використанням нейронних мереж та дерев ухвалення рішень [59]. Залежно від завдання, яке вирішується, необхідно оцінити ймовірність виникнення аномальної події та оцінити її масштаб і відхилення від нормальних значень. В запропонованому в роботі [60] методі використовується ієрархічний Pattern Matching (PM), що надає можливість наближено оцінити характер аномалії. В роботі [61] зіставлення шаблонів використовується не для пошуку аномалій, а для точної оцінки масштабу неповторюваних подій у реальному часі.

Окрім розглянутих методів прогнозування часових рядів необхідно звернути увагу на методи на базі нейронних мереж та РМ. Використання нейронних мереж дозволяє виявляти складні залежності в історичних даних, що теоретично може дозволити прогнозувати аномальні навантаження. Проте використання даного підходу несе відповідні ризики, оскільки пояснення прийнятих рішень нейронною мережею є нетривіальною задачею. Методи на

основі РМ базуються на пошуку схожих ситуацій у минулому та екстраполяції на поточний стан. Якщо повернутися до припущення, що аномальні навантаження мають обмежену кількість причин, то можна зробити висновок, що і шаблони навантаження будуть повторюватися.

#### 2.4.1 Розробка методу комбінованого прогнозування

РМ є методом прогнозування часових рядів, що базується на пошуку найбільш подібних шаблонів в історичних даних та їх екстраполяції до поточного стану [61]. Цей метод ефективний для виявлення і прогнозування поведінкових шаблонів, що мають тенденцію до повторювання без постійної періодичності.

Для визначення схожості шаблонів в часових рядах в РМ використовуються різні метрики відстані, наприклад, Евклідова відстань або Dynamic Time Warping. Дані метрики дозволяють оцінити наскільки один сегмент схожий на інший. Наприклад, Евклідова відстань є найпростішим методом оцінити подібність двох часових рядів:

$$d(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}, \quad (2.4)$$

де  $X=[x_1, x_2, \dots, x_n]$  та  $Y=[y_1, y_2, \dots, y_n]$  – часові ряди, які порівнюються, а  $d(X, Y)$  – функція дистанції.

Метод Dynamic Time Warping є розширенням Евклідової відстані, що дозволяє порівнювати часові ряди різної швидкості за часом [62]:

$$DTW(X, Y) = \sqrt{\sum_{(i,j) \in W} (x_i - y_j)^2}, \quad (2.5)$$

де  $W(k)$  – функція вирівнювання, яка визначає зміщення часових рядів  $X$  та  $Y$  один відносно іншого.

Для пошуку необхідного шаблону за допомогою обраної функції відстані можуть використовуватися різні підходи. Часовий ряд розбивається на сегменти встановленої довжини  $Q$  і при пошуку порівнюються цільовий шаблон і кожен з сегментів:

$$k = \arg \min_i d(Q, C_i), \quad (2.6)$$

де  $k$  – індекс найбільш подібного сегменту,  $d(Q, C_i)$  – функція дистанції між часовими рядами,  $C_i$  –  $i$ -й часовий ряд. На основі знайденого шаблону навантаження виконується екстраполяція, що є безпосередньо процесом прогнозування.

Описана модель демонструє основні недоліки даного підходу прогнозування. Повторювані шаблони можуть мати різну тривалість, а сегментованість історичного часового ряду є сталою, що не дозволяє прогнозувати тривалі шаблони. Крім того, даний підхід є неефективним, якщо історичні дані не містять подібних шаблонів. Проте основна перевага перед іншими методами прогнозування є адаптивність до нетипових шаблонів, які вже присутні в історичних даних. Дана властивість є критично важливою для покращення стійкості до аномалій існуючих методів прогнозування.

Розглянуті методи прогнозування, зокрема, Prophet, здатні працювати з комплексними сезонностями, трендами та вказаними подіями, але не є стійкими до аномалій. Одним із можливий варіантів покращення даної характеристики є комбінування Prophet та РМ. Комбінування методів прогнозування заключається в лінійній або нелінійній комбінації прогнозів кожного задіяного методу. При лінійному комбінуванні робота методів є незалежною від інших. Входом в такому випадку є тільки історичні дані і безпосередньо параметри методів. Прогнози всіх методів далі комбінуються, зокрема, методом вагових коефіцієнтів, що зображено на рис. 2.7:

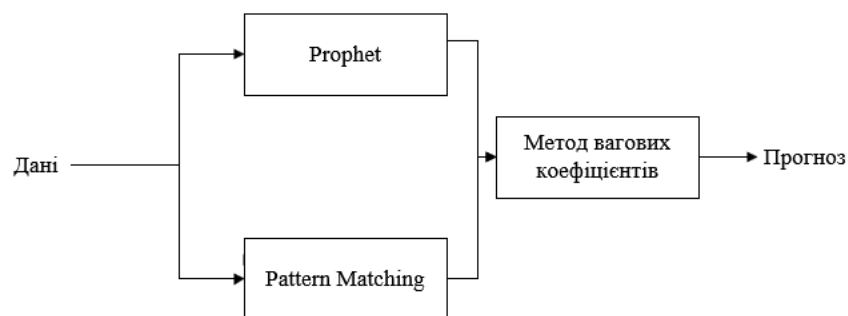


Рисунок 2.7 – Лінійне комбінування методів прогнозування

В нелінійній комбінації результати роботи одного або кількох методів є вхідними даними для роботи іншого, що дозволяє використовувати нелінійні залежності під час прогнозування. Бібліотека Prophet крім прогнозування майбутніх значень навантаження, також надає додаткову інформацію про сезонності, тренди і аномалії в історичних даних, що може бути використано для покращення результатів РМ:

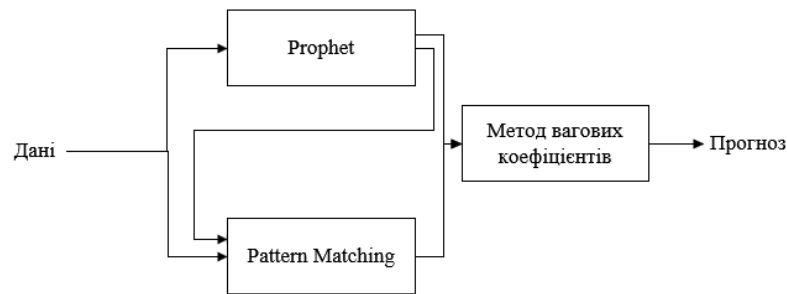


Рисунок 2.8 – Нелінійне комбінування методів прогнозування

В роботі розглядається перший варіант, що наведено на рис. 2.7, оскільки метою дослідження є доведення можливості комбінування двох методів для покращення окремих характеристик кожного з них. Нелінійне комбінування буде використано для оптимізації методу, що розробляється.

Маючи прогнози двох методів необхідно скомбінувати їх для подальшої роботи проактивних методів масштабування. Для цього можна використати різні методи комбінації часових рядів, зокрема, метод вибору найкращої моделі, метод середнього, метод вагових коефіцієнтів. Тоді задача оптимізації виглядає наступним чином:

$$\begin{cases} \hat{y}_{\text{комбінований}}(t) = \alpha \cdot \hat{y}_{\text{Prophet}}(t) + \beta \cdot \hat{y}_{\text{PM}}(t), \\ \alpha + \beta = 1, \end{cases} \quad (2.7)$$

де  $\hat{y}_{\text{комбінований}}(t)$  є функцією прогнозу комбінованого методу, а  $\alpha$  та  $\beta$  ваговими коефіцієнтами для Prophet та РМ відповідно.

Тоді цільова функція, яку необхідно оптимізувати виглядає наступним чином:

$$J(\alpha, \beta) = \frac{1}{N} \sum_{i=1}^N (y(t_i) - (\alpha \cdot \hat{y}_{\text{Prophet}}(t_i) + \beta \cdot \hat{y}_{\text{PM}}(t_i)))^2, \quad (2.8)$$

де  $J(\alpha, \beta)$  – цільова функція, параметри якої необхідно оптимізувати,  $N$  – кількість точок для перевірки точності, а  $t_i$  – час для  $i$ -тої точки.

$$(\alpha^*, \beta^*) = \arg \min_{\alpha, \beta} (J(\alpha, \beta)), \quad (2.9)$$

де  $\alpha^*$  та  $\beta^*$  – вагові коефіцієнти при яких цільова функція набуває найменшого значення, що відповідає найкращій точності. Для оптимізації можна використати метод перебору, метод градієнтного спуску або послідовного квадратичного програмування. З використанням обмеження

$$\begin{cases} \alpha + \beta = 1, \\ 0 \leq \alpha \leq 1, \\ 0 \leq \beta \leq 1, \end{cases} \quad (2.10)$$

де  $\alpha$  та  $\beta$  – вагові коефіцієнти, але теоретично кількість методів і коефіцієнтів може зрости, з'являється можливість використовувати послідовне квадратичне програмування, що полягає в пошуку оптимальних значень методом градієнтного спуску з врахуванням встановлених обмежень [63]:

$$\begin{cases} \alpha_{k+1} = \alpha_k - LR \frac{\partial J}{\partial \alpha}, \\ \beta_{k+1} = \beta_k - LR \frac{\partial J}{\partial \beta}, \end{cases} \quad (2.11)$$

де  $\alpha_{k+1}$  та  $\beta_{k+1}$  – значення коефіцієнтів на наступному кроці пошуку,  $LR$  – параметр швидкості пошуку. Похідна цільової функції в даній формулі обчислюється наступним чином:

$$\begin{cases} \frac{\partial J}{\partial \alpha} = -\frac{2}{N} \sum_{i=1}^N (y(t_i) - (\alpha \cdot \hat{y}_{\text{Prophet}}(t_i) + \beta \cdot \hat{y}_{PM}(t_i))) \cdot \hat{y}_{\text{Prophet}}(t_i), \\ \frac{\partial J}{\partial \beta} = -\frac{2}{N} \sum_{i=1}^N (y(t_i) - (\alpha \cdot \hat{y}_{\text{Prophet}}(t_i) + \beta \cdot \hat{y}_{PM}(t_i))) \cdot \hat{y}_{PM}(t_i). \end{cases} \quad (2.12)$$

Використання послідовного квадратичного програмування в такому вигляді дозволяє за обмежений час знайти оптимальні значення вагових коефіцієнтів  $\alpha$  та  $\beta$  для застосування в (2.7).

### 2.4.2 Дослідження комбінованого методу

Для перевірки працездатності запропонованого комбінованого методу проводиться тестування на синтетичних даних. Кожен експеримент перевіряє окрему ситуацію навантаження та проводиться порівняння комбінованого та методів на базі Prophet і РМ. Для використання РМ методу в мові програмування Python обрано бібліотеку Stumpy, яка має вбудований інтерфейс для аналізу часових рядів. Для комбінування результатів використовується бібліотека Scipy, яка надає реалізацію послідовного квадратичного програмування.

Перший експеримент націлений на базову перевірку адекватності. Згенеровані дані для експерименту містять часовий ряд з нетиповим навантаженням протягом одного періоду сезонності. Зеленим кольором зображено дані, які надаються для тренування моделей, а жовтим позначені умовно майбутні значення навантаження.

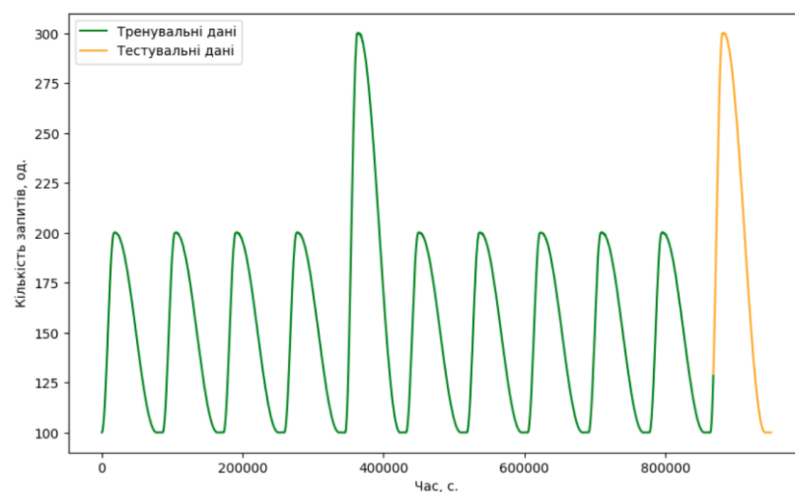


Рисунок 2.9 – Шаблон навантаження для базової перевірки

На рис. 2.10 зображені результати прогнозування. Оскільки дані для перевірки містить аномальне навантаження, яке вже міститься в історичних даних, то очікується, що метод РМ має бути точнішим, а комбінований повністю його повторювати, що і демонструється на рис. 2.10.

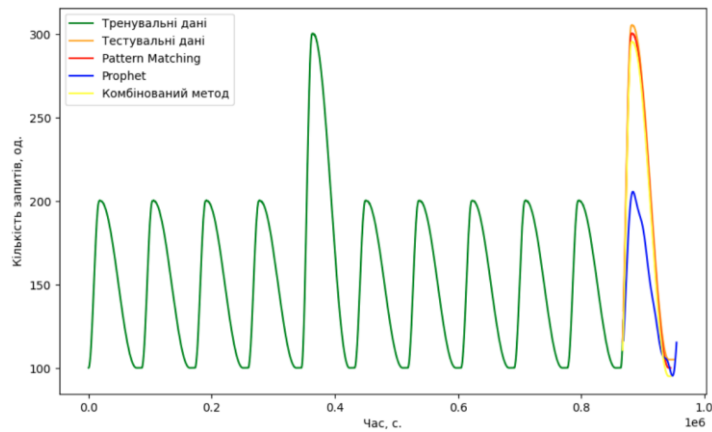


Рисунок 2.10 – Результати прогнозування під час базової перевірки

Наступний експеримент проводиться на згенерованих даних, які містять постійний шум та незначні відхилення від загального шаблону коливань. Наявність шуму в даних наближає умови експерименту до реальних. Дані містять просту сезонність та не містять тенденцій, коливань або аномалій. Шаблон навантаження зображений на рис. 2.11. Дані, що використовуються для навчання, містять частину валідаційного періоду, що необхідно для коректної роботи РМ.

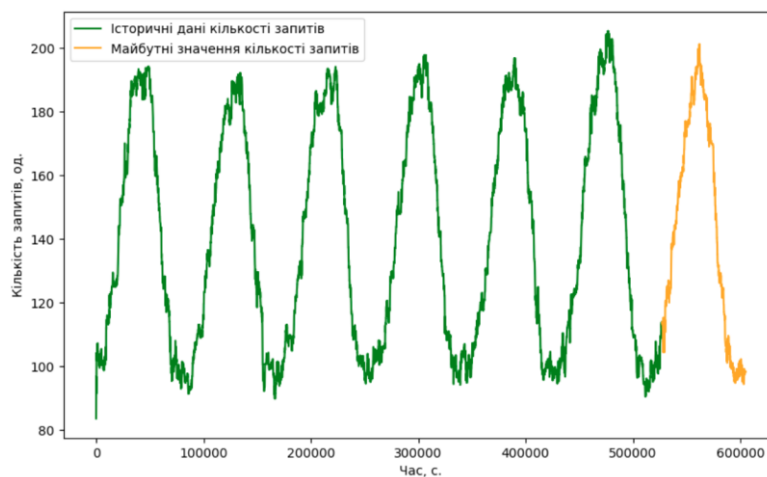


Рисунок 2.11 – Шаблон навантаження з простою сезонністю

На рис. 2.12 зображено результати прогнозування для часового ряду з простою сезонністю. Оскільки історичні дані не містять аномалій та мають чітку сезонність, точність прогнозів є високою.



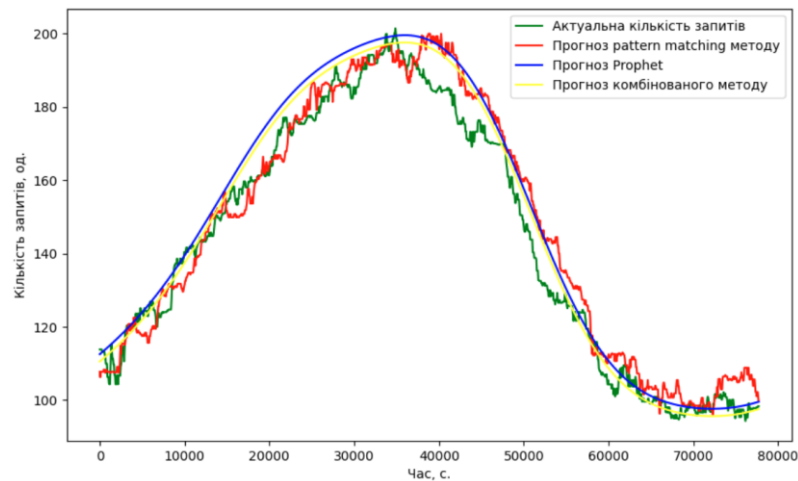


Рисунок 2.12 – Результати прогнозування на шаблоні з простою сезонністю

Наступний експеримент окрім простої сезонності також містить аномальні навантаження. Прогнозування відбувається саме для періоду з аномалією. Аномальне навантаження присутнє в п'ятому періоді історичних даних та міститься в даних для валідації, що дозволить перевірити застосування компоненти короткострокового прогнозування.

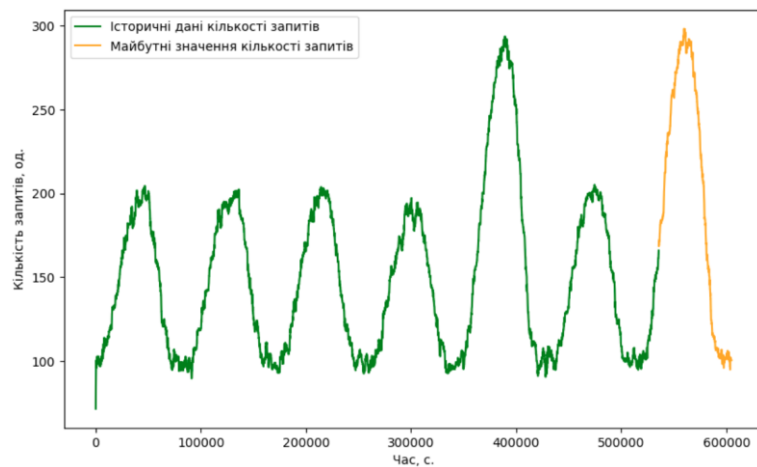


Рисунок 2.13 – Шаблон навантаження з простою сезонністю та аномаліями

На рис. 2.14 зображено результати прогнозування для часового ряду з простою сезонністю та аномаліями. Очікувано, що РМ демонструє кращу точність, оскільки шаблон аномального навантаження вже міститься в історичних даних. Оскільки наявне аномальне навантаження не є сезонним, Prophet показав низьку точність. Відповідно, комбінований метод використовує саме результати РМ.

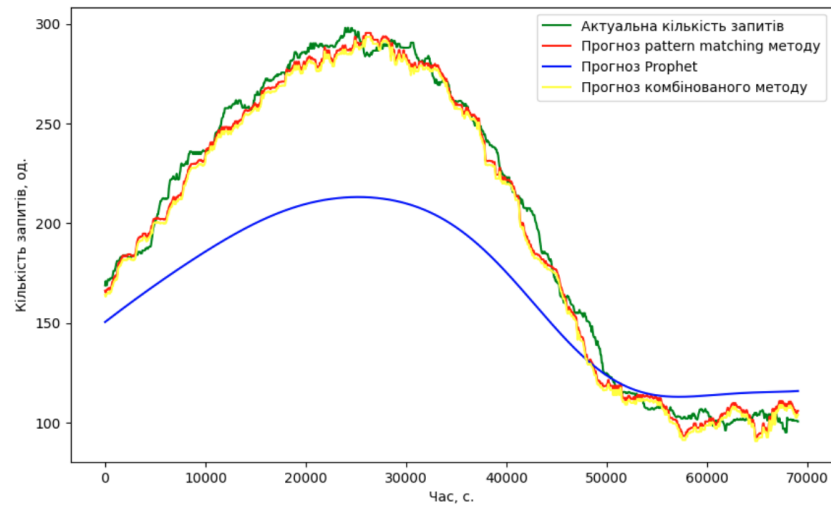


Рисунок 2.14 – Результати прогнозування на шаблоні з простою сезонністю та аномаліями

В наступному експерименті, що зображений на рис. 2.15, перевіряється робота на даних з простою сезонністю, зростаючою тенденцією і аномаліями в п'ятому і шостому (валідаційному) періоді навантаження. Описаний експеримент дозволяє перевірити здатність РМ масштабувати знайдені шаблони до поточних умов.

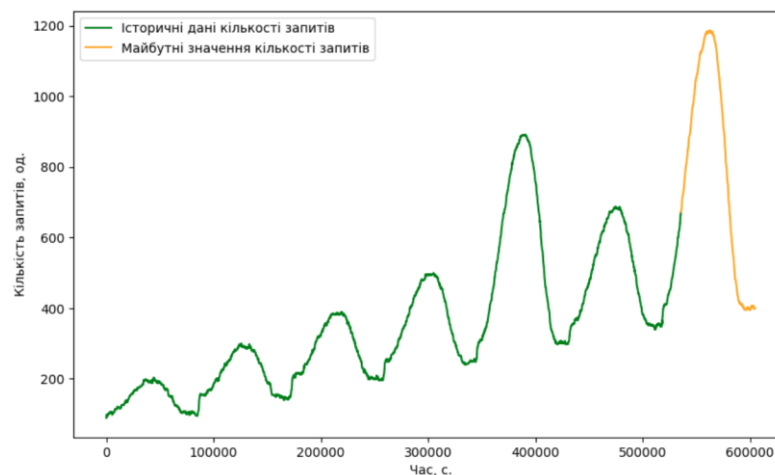


Рисунок 2.15 – Шаблон навантаження з простою сезонністю, аномаліями та зростаючим трендом

В даному експерименті через наявність аномального навантаження Prophet є недостатньо точним. Оскільки шаблон аномального навантаження вже міститься в історичних даних, РМ масштабував ідентифікований шаблон

для адаптації до зростаючого тренду. Комбінований метод аналогічно до попереднього експерименту використовує прогноз РМ.

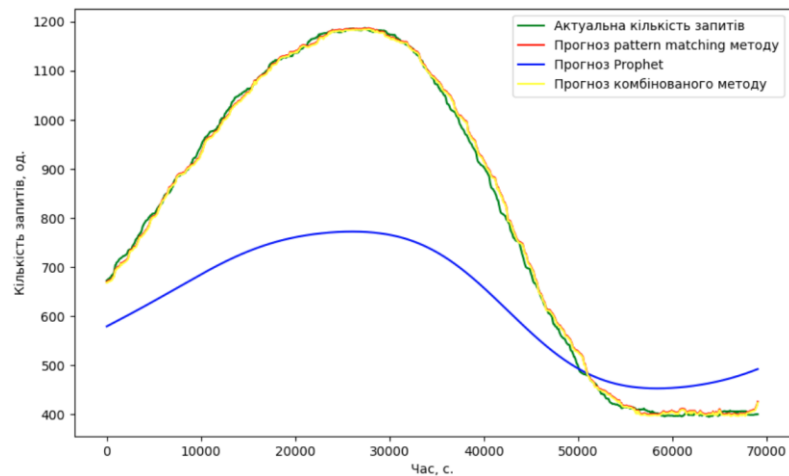


Рисунок 2.16 – Результати прогнозування на шаблоні з простою сезонністю, аномаліями та зростаючим трендом

Наступний експеримент виконується на даних з комплексною сезонністю. Крім того, тренувальні дані не містять дані з наступного періоду сезонності, що не дозволяє РМ знайти необхідний шаблон. Даний експеримент демонструє перевагу саме довгострокового прогнозування.

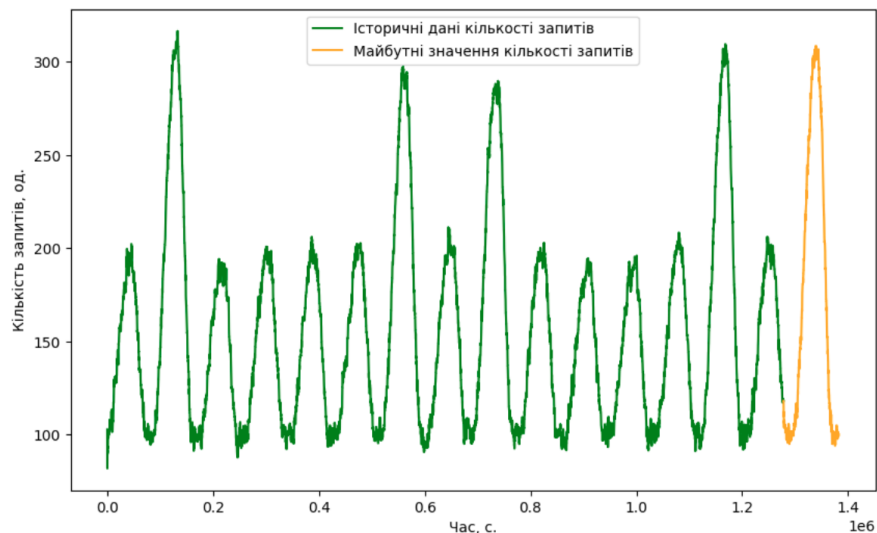


Рисунок 2.17 – Шаблон навантаження з комплексною сезонністю

Через нестачу даних для розпізнавання необхідного шаблону РМ метод демонструє недостатню точність, що зображено на рис 2.18. В той же час,

Prophet, який оперує сезонностями, завчасно надає прогнози високої точності, які є базовими для комбінованого методу.

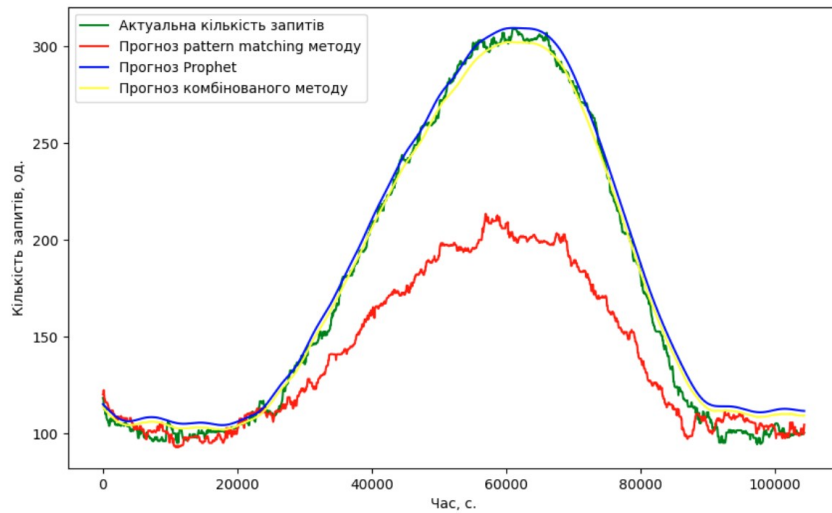


Рисунок 2.18 – Результати прогнозування на шаблоні з комплексною сезонністю

Останній експеримент дозволяє оцінити ефективність комбінованого методу протягом семи валідаційних періодів. Дані є синтетичними та містять комплексну сезонність – денну та тижневу, що зображено на рис. 2.18. В історичних даних містяться аномалії, що відтворюють однаковий шаблон, та не мають константного періоду повторення. Прогнозування відбувається на початку кожного періоду денної сезонності.

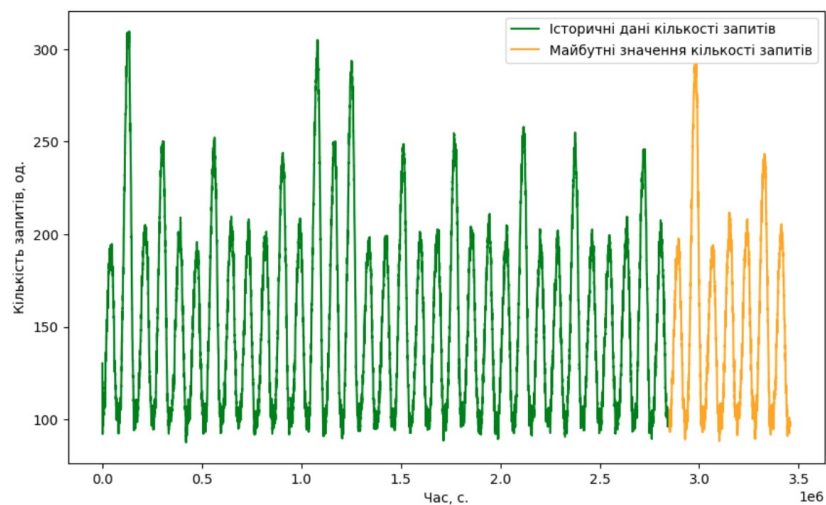


Рисунок 2.18 – Дані для оцінки точності розроблених методів

Результати прогнозування зображені на рис. 2.20. Шаблон навантаження першого періоду не містить аномалій, тому Prophet надав прогноз високої точності і є базовим для комбінованого методу. На другому періоді міститься аномалія, яка є коректно ідентифікована РМ. Інші періоди аномалій не містять. На шостому періоді є тижневе збільшення навантаження, для якого Prophet надав коректний прогноз.

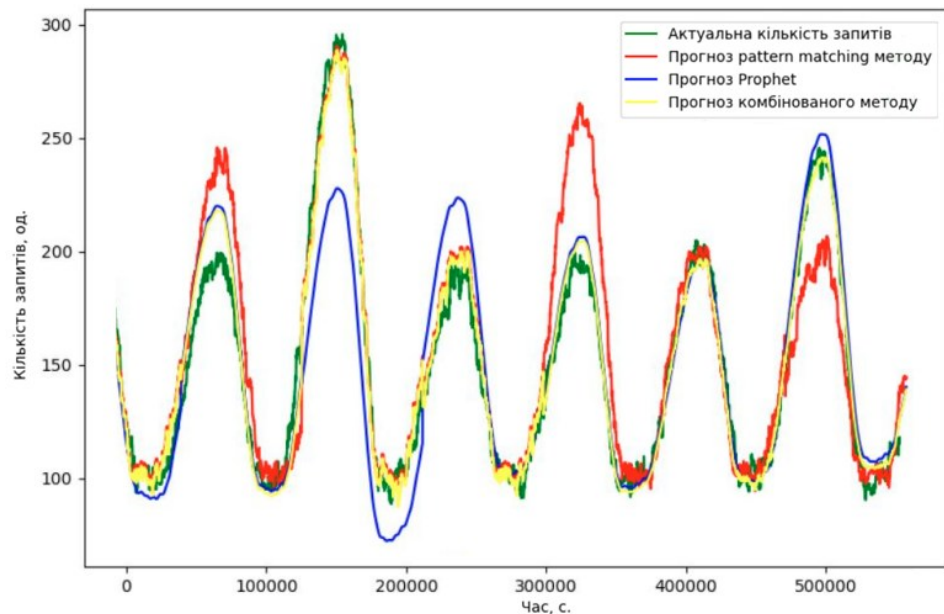


Рисунок 2.20 – Результати тестування розроблених методів

В таблиці 2.6 наводяться метрики точності MAPE отримані в результаті експериментів. Використання комбінованого методу підвищує ефективність прогнозування з 91.32% до 95.4% у порівнянні з Prophet, що доводить ефективність запропонованого методу.

Таблиця 2.6 – Метрики точності для методів прогнозування

Метод прогнозування	MAPE, %
РМ	9.23
Prophet	8.68
Комбінований	4.60

Проведені експерименти доводять, що використання комбінації методів прогнозування РМ і Prophet можна використовувати задля підвищення

стійкості прогнозування до аномальних навантажень. Експерименти проведені на згенерованих даних, тому в майбутніх дослідженнях комбінованого методу необхідно включити дані реальних навантажень для більш точної оцінки релевантності запропонованого методу. Точність запропонованого методу безпосередньо залежить від точності методів, що використовуються при комбінуванні.

## 2.5 Розробка проактивного методу масштабування

### 2.5.1 Сутність проактивного методу масштабування

Наявність точного та стійкого до аномалій методу прогнозування є основною умовою для побудови методу проактивного масштабування. Для розробки даного методу необхідно визначити основну рівність при управлінні ОР:

$$k_l(t) \cdot X_{ly}(t) = W_{ly}(t + D_{up}), l = \overline{1, m}, y = \overline{1, h}, \quad (2.13)$$

де  $W_{ly}(t)$  – функція навантаження на  $l$ -й застосунок для  $y$ -го ОР, яка є невідомою для майбутніх часових проміжків, але методи прогнозування дозволяють отримати її апроксимацію. Функція  $k_l(t)$  описує кількість екземплярів  $l$ -го застосунку в залежності від часу, а  $X_{ly}(t)$  є функцією об'єму запитів на  $y$ -ий ресурс  $l$ -го застосунку. Параметр визначає затримку розгортання нових екземплярів. Таким чином рівність (2.13) визначає, що загальний об'єм ОР має бути еквівалентний потребам з врахуванням часового проміжку. При  $D_{up} = 0$  дана рівність є справедливою для реактивного масштабування.

Проактивне масштабування може відбуватися як горизонтально, так і вертикально. Розглянемо перший випадок, при якому функція запитів ОР є константою. Тоді функція кількості екземплярів визначається як:

$$k_l(t) = \max\left(\left\lceil \frac{W_{ly}(t + D_{up})}{X_{ly}} \right\rceil\right), l = \overline{1, m}, y = \overline{1, h}. \quad (2.14)$$

Ця функція описує залежність кількості екземплярів  $l$ -го застосунку від часу враховуючи неоднорідність завантаженості всіх доступних ресурсів і беручи до уваги найбільшу потребу. Кількість екземплярів має бути цілим, тому відбувається округлення вгору для забезпечення необхідного об'єму.

У випадку вертикального масштабування константою є кількість екземплярів  $k_l(t)$ , а функція об'єму  $y$ -го ресурсу визначається як:

$$X_{ly}(t) = \left\lceil \frac{W_{ly}(t + D_{up})}{k_l} \right\rceil, l = \overline{1, m}, y = \overline{1, h}. \quad (2.15)$$

Запит визначається для всіх екземплярів одночасно, оскільки береться до уваги роботи балансувальника навантаження. Кожен ресурс є незалежним та визначається індивідуально.

Оскільки рівності наведені вище не є дискретними, а їх обрахунок займає деякий час та відбувається із заданою періодичністю, то необхідно врахувати проміжні значення. Для цього вводиться параметр кроку  $t_{step}$  для дискретизації описаних формул. В даному методі пропонується враховувати максимальне значення функції навантаження  $W_{ly}(t)$  на часовому проміжку від  $t$  до  $t + D_{up}$ :

$$k_l(t) = \max \left( \left\lceil \frac{\max(W_{ly}(t + t_{step}))}{X_{ly} \cdot \varepsilon_y} \right\rceil \right), l = \overline{1, m}, y = \overline{1, h}, t_{step} = \overline{1, D_{up}}. \quad (2.16)$$

Аналогічна трансформація застосовується для вертикального масштабування. Крім того, необхідно врахувати коефіцієнт запасу  $\varepsilon_y$  для випадків з нетиповим навантаженням:

$$X_{ly}(t) = \left\lceil \frac{\max(W_{ly}(t + t_{step}))}{k_l \cdot \varepsilon_y} \right\rceil, l = \overline{1, m}, y = \overline{1, h}, t_{step} = \overline{1, D_{up}}. \quad (2.17)$$

Тоді можемо визначити умови масштабування для горизонтального масштабування:

$$k_l(t + 1) = k(t) + \Delta k, k_l(t + D_{up}) > k_l(t), l = \overline{1, m}, \quad (2.18)$$

Для вертикального масштабування використовується аналогічний підхід, але значення об'єму для кожного типу ОР обчислюється окремо:

$$X_{ly}(t+1) = X_{ly}(t) + \Delta X_{ly}, X_{ly}(t + D_{up}) > X_{ly}(t), l = \overline{1, m}, y = \overline{1, h}. \quad (2.19)$$

Необхідно також врахувати, що в контексті одного розгортання застосунку вертикальна конфігурація встановлюється для всіх екземплярів, відповідно необхідно враховувати використання ОР всіма доступними екземплярами застосунку.

### 2.5.2 Особливості реалізації проактивного методу масштабування

Реалізація запропонованого методу проактивного масштабування відбувається за допомогою окремих модулів, кожен з яких має свою окрему функцію. Модуль даних отримує дані від джерела історичних даних та здійснює їх оброблення. Модуль прогнозування прогнозує майбутні навантаження. Модуль валідації оцінює доцільність застосування отриманих прогнозів. Модуль застосування взаємодіє через API з кластером з метою застосування отриманих значень об'ємів ОР у разі необхідності.



Рисунок 2.21 – Діаграма компонентів проактивного методу масштабування



Коли рішення, яке реалізує описаний метод, ініціалізується для автоматизації масштабування цільового застосунку, попередні метрики використання можуть бути відсутні. У такому випадку, дане рішення встановлює визначену адміністратором кількість реплік `defaultReplicas` доти, поки не буде отримано достовірний прогноз. Достовірність прогнозу використання ресурсів визначається за допомогою показника `MARE`. Якщо помилка моделі є меншою за параметр впевненості, тоді даний метод масштабування, покладаючись на отримані значення, встановлює розраховано оптимальну кількість реплік змінюючи поле `replicas` в Kubernetes маніфесті розгортання [10].

Історичні дані отримуються шляхом запитів на Prometheus сервер. Для отримання історичних метрик використання CPU використовується запит вигляду `sum(rate(container_cpu_usage_total {container!=""}[60s])) by (pod)`. Для отримання поточних заданих запитів використовується API Kubernetes, а саме поле `spec.container[0].requests.cpu` в маніфесті розгортання.

Обраний алгоритм прогнозування має автоматично виявляти сезонності, тренди та аномалії в часових рядах, тому не потребує додаткового налаштування. Проте адміністратор може вказати базову сезонність `seasonalities` для покращення точності прогнозів. Навчання та оцінка точності моделі відбувається з заданим періодом `trainEvaluatePeriod`. При падінні показників точності моделі, рішення встановлює кількість реплік `defaultReplicas`, до моменту поки не буде отримана необхідна точність.

При розробці рішення для горизонтального масштабування необхідно врахувати, що застосунки потребують деякий час на ініціалізацію. Наприклад, у випадку бази даних Redis, застосунку необхідно розгорнути збережений стан бази даних та встановити з'єднання з іншими компонентами кластера. Крім того, після команди на збільшення кількості реплік застосунку, потребується деякий час на ініціалізацію поду, а саме розгортання його на доступному вузлі, завантаження образу та під'єднання мережових дисків. Тому рішення має

додатковий параметр `applicationTimeToStart` для того, аби точно розрахувати момент, коли необхідно збільшити кількість реплік застосунку.

Відповідно, рішення перевіряє необхідність масштабувати застосунок з коротким інтервалом. При масштабуванні вгору, перевіряється прогнозовані значення використання в момент часу `currentTime + applicationTimeToStart`, аби встановити необхідну кількість реплік завчасно і не допустити падіння метрик якості сервісу. При масштабуванні вниз, перевіряється лише значення в поточний момент, щоб не зменшити кількість реплік завчасно. Тому маємо спрощено формулу для обчислення необхідної кількості реплік на основі прогнозу –  $\max(\text{forecastedUsage}[\text{currentTime} + \text{timeToStart}], \text{actualUsage}[\text{currentTime}])$ .

Запропоноване рішення розміщується в кластері Kubernetes, що зображенку на рис. 2.22, і має безпосередній доступ до Prometheus сервера та API Kubernetes.

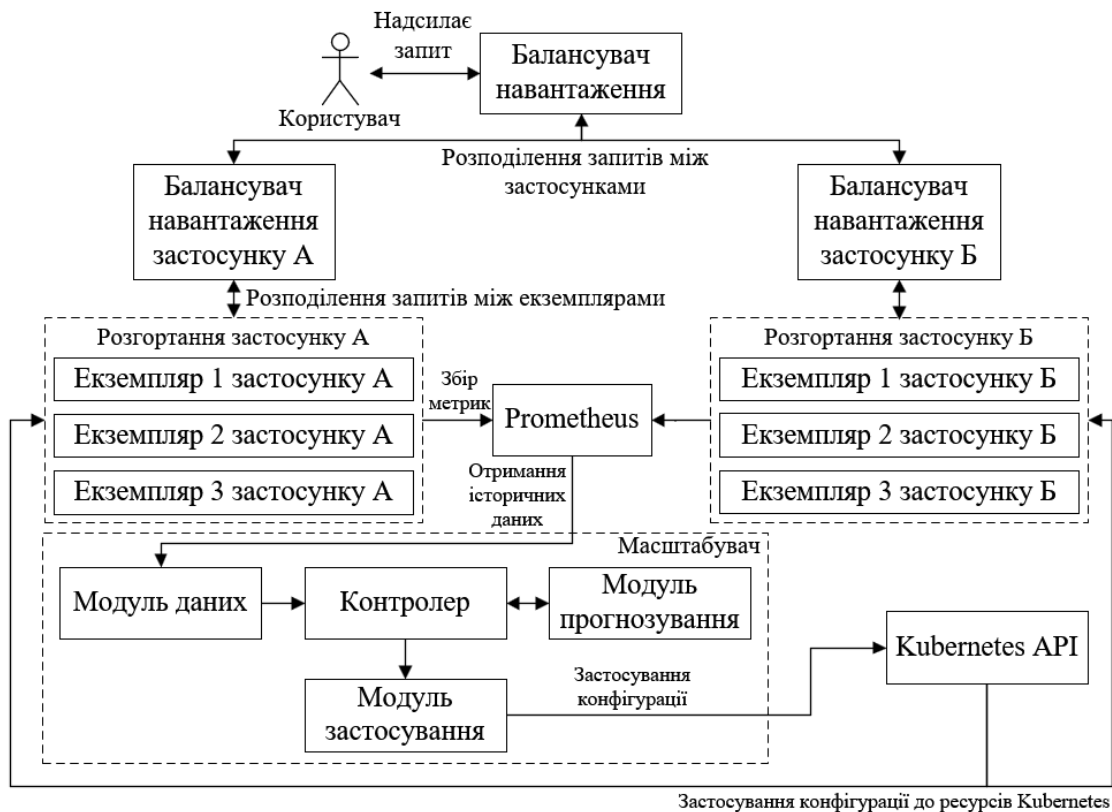


Рисунок 2.22 – Схема інтеграції масштабувача в Kubernetes

### 2.5.3 Дослідження проактивного методу

Для тестування отриманого рішення і його порівняння з іншими підходами використовується кластер Kubernetes на базі minikube. Кластер включає єдиний вузол, який має шість процесорних ядер з тактовою частотою 3.6GHz та 16 гігабайт оперативної пам'яті. Мережева комунікація не виходить за рамки однієї машини. Цей кластер містить встановлений Prometheus для моніторингу за допомогою kube-prometheus stack.

Для перевірки обрано сценарій з періодичним навантаженням. Період навантаження становить 300 с. Сумарне мінімальне навантаження становить 100 мілікорів, максимальне – 550 мілікорів.

Навантажувальне тестування здійснюється за допомогою спеціалізованої утиліти locust для надсилання HTTP запитів, екземпляр якої також розміщується в кластері. Схема експерименту зображена на рис. 2.23. Навантажувальний шаблон запитів складається з періодичних коливань. Період коливань становить 300 с, мінімальна частота запитів – 10 запитів в секунду, максимальна – 50.

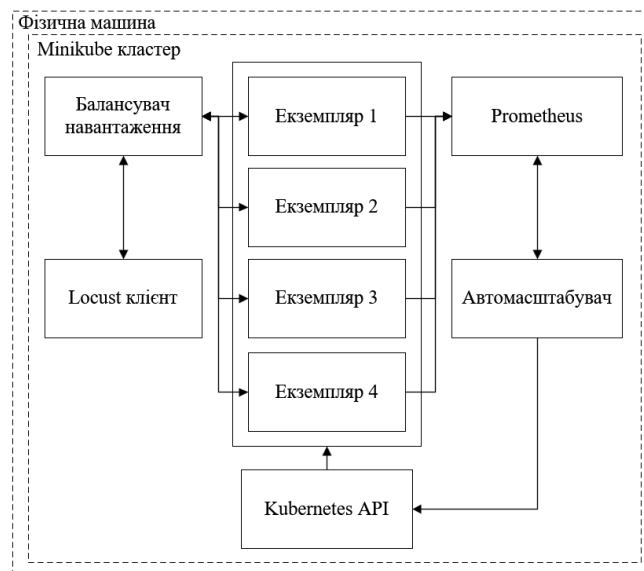


Рисунок 2.23 – Конфігурація експериментів

Тестовий застосунок є веб-сервером, який на кожен запит виконує деяку CPU-інтенсивну роботу. Такий тестовий застосунок може бути масштабованим як горизонтально, так і вертикально. Кількість реплік

застосунок або об'єм виділених ресурсів не впливають на роботу застосунку. Застосунок ініціалізується заданий час перед тим, як стати доступним для запитів і перед тим, як перевірки доступності Kubernetes будуть вважатися успішними. В проведених експериментах цей час дорівнює п'яти секундам.

В даному експерименті масштабування відбувається лише для CPU ресурсу. Запит на процесорний ресурс для тестового застосунку становить 200 мілікорів. Оскільки при перевищенні даного значення до застосунку буде застосований тротлінг, що може вплинути на результати тестування, встановлено ліміт на ресурс процесорного часу, що дорівнює 250 мілікорів. Початкова кількість реплік дорівнює п'яти.

HPA – вбудоване рішення для автоматизації горизонтального масштабування в Kubernetes. HPA є представником реактивного підходу, тому не містить жодних алгоритмів прогнозування. Ідея HPA полягає в тому, аби за допомогою регуляції кількості реплік підтримувати встановлене адміністратором значення середнього навантаження на екземпляр – `targetAverageUtilization`. Основна різниця полягає в тому, що рішення про масштабування приймається на основі поточних значень використання ОР.

Оскільки реактивний підхід сильно залежить від затримок отримання поточних даних, необхідно мінімізувати фактор затримок на результати експерименту. HPA використовує `metrics-server` як джерело даних, тому налаштування `resolution` для `metrics-server` встановлене в мінімально можливе значення – 15 с. Також встановлене мінімальне значення `sync-period`, `cpu-initialization-period`, `initial-readiness-delay` для пришвидшення часу реакції HPA на зміни. Крім цього, в політиках масштабування `behavior` знято ліміти на швидкість масштабування вниз та вгору.

На рис. 2.24 зображено результати проведення тестового сценарію без використання автоматичного масштабування. Дана пара графіків містить дані про фактичне сумарне використання, резервування процесорного часу і результуючий час обробки запитів. В наведених умовах застосунок має достатню кількість екземплярів для обробки отриманих запитів. При

максимальному навантаженні сумарне фактичне використання процесорного часу становить 570 мілікорів. При цьому час відповіді в пікові моменти зростає з 12 мс до 90 мс, що фактично є показником QoS в даному експерименті.

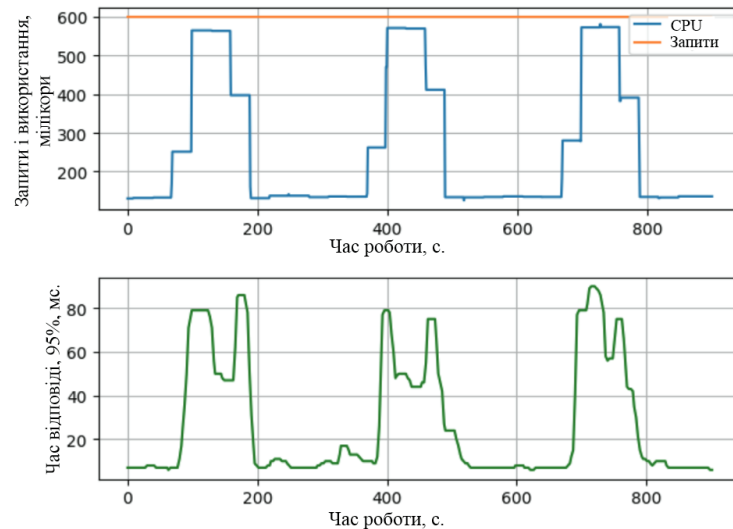


Рисунок 2.24 – Результати роботи застосунку зі статичними запитами

На рис. 2.25 зображено результати тестування розробленого рішення. Даний графік демонструє, що час відповіді тестового застосунку є аналогічним першому експерименту – від 12 мс до 90 мс, тому така поведінка може трактуватися як особливість роботи застосунку. Масштабування вгору відбувалося завчасно, а масштабування вниз – після проходження пікового навантаження.

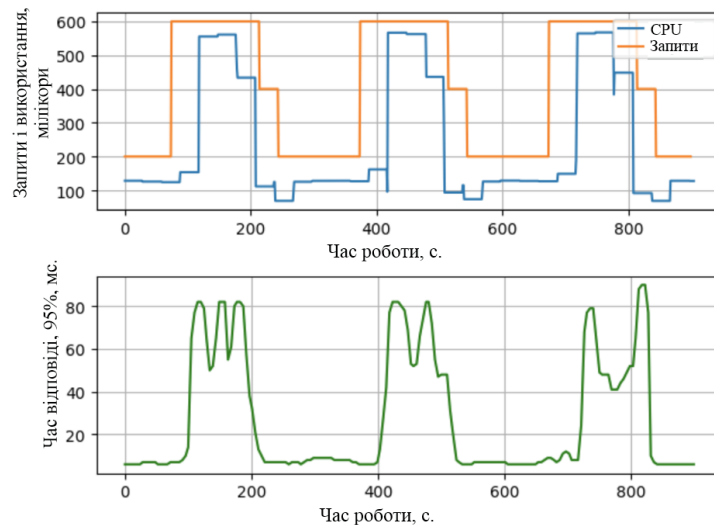


Рисунок 2.25 – Результати роботи з проактивним масштабуванням

На рис. 2.26 зображено результати тестування НРА з описаною раніше конфігурацією. Необхідно зазначити, що метрика по зарезервованим ресурсам є сумою запитів всіх подів, які знаходяться в статусі Ready. Тому на рис. 2.26 можна побачити, що є невелика затримка між збільшенням фактичного використання ресурсу і наданим ресурсом. Через це виникає короткочасне погіршення QoS в момент затримки масштабування, а саме збільшення часу відповіді з 90 мс в попередніх експериментах до 1700 мс.

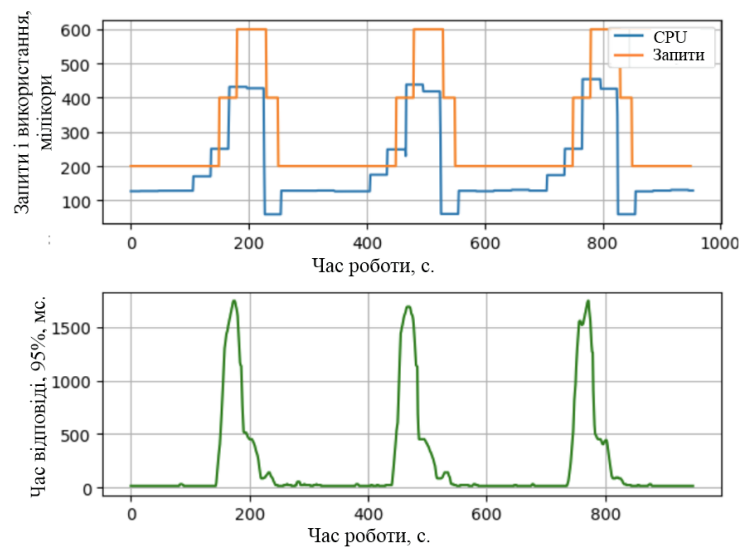


Рисунок 2.26 – Результат роботи з НРА

На рис. 2.27 порівнюються метрики часу відповіді для всіх трьох конфігурацій і можливо оцінити масштаб різниці між роботою методів, що розглядаються.

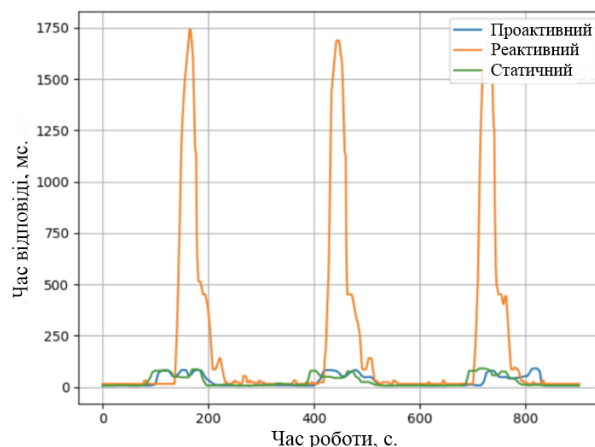


Рисунок 2.27. Порівняння часу відповіді всіх обраних підходів

Таблиця 2.7 містить дані про середній час відповіді, а також 95 та 99 перцентилі. Зокрема, в середньому всі значення часу відповіді для проактивного і реактивного підходів відрізняються в вісім разів на користь першого, але резервує на 26% більше ресурсу. Також показники часу відповіді для рішення, що пропонується, і управління без масштабування є аналогічними, але при цьому резервування ресурсу зменшується на 47%. Варто зазначити, що умови проведеного експерименту спрямовані на демонстрацію переваги проактивного підходу при швидкозростаючих шаблонах навантаження, що є значною проблемою реактивного підходу.

Таблиця 2.7 – Порівняння результатів роботи

Підхід	Середній, мс	95%, мс	99%, мс
Проактивний	23	87	190
Статичний	22	88	188
Реактивний	160	770	1350

Отримані результати свідчать про працездатність та ефективність розробленого рішення. Експерименти проведені для процесорного часу. Іншим ключовим ОР є пам'ять. Пам'ять має свою специфіку виділення ресурсів, оскільки при перевищенні встановлених запитів або лімітів, застосунок може бути аварійно завершений. Крім того, на відміну від процесорного часу, деяка частина пам'яті використовується незалежно від навантаження – для зберігання коду і початкових даних для роботи. Це означає, що розрахунок сумарної кількості пам'яті при горизонтальному масштабування має включати описану вище постійну складову.

## Висновки до розділу 2

В розділі визначені недоліки реактивного масштабування та обґрунтовано необхідність розробки проактивного рішення для управління ОР. Визначено, що ключовою частиною проактивних рішень є методи

прогнозування робочих навантажень або використання ресурсів. Проаналізовано існуючі методи, що здатні працювати з сезонними навантаженнями та тенденціями. На основі експериментальних досліджень обрано множину таких методів, що можуть бути використані для подальшої розробки проактивних методів. Обґрунтовано вибір метода прогнозування Prophet.

Запропоновано метод комбінованого прогнозування, що включає компоненти для короткострокового та довгострокового прогнозування. Доведено, що поєднання РМ та Prophet дозволяє отримати стійкий до аномальних навантажень метод прогнозування. Експериментальні дослідження показали зростання точності прогнозування з 91% до 95% у порівнянні з базовими методами на згенерованих даних.

Запропоновано метод проактивного масштабування. Формалізовано модель проактивного управління. На основі моделі визначено основні модулі, зокрема, модуль даних, аналізу, валідації і прийняття рішень. Експериментальні дослідження показали ефективність розробленого методу при наявності комплексних сезонностей та тенденцій в часових рядах. Було отримано зменшення середнього часу відповіді з 160 мс до 23 мс при збільшенні використання ресурсів на 26%. У порівнянні зі статичними запитами отримано аналогічний рівень якості послуг при зменшенні об'ємів зарезервованих ресурсів на 47%.



### **3 РОЗРОБКА ГІБРИДНИХ МЕТОДІВ МАСШТАБУВАННЯ**

В третьому розділі пропонується покращення запропонованого методу проактивного горизонтального масштабування. Проактивні методи є ефективними при наявності точних прогнозів, але при появі аномалій в навантаженні виникає проблема некоректного розподілу ресурсів і в результаті – порушення встановлених SLA. В розділі пропонується гібридний метод, який дозволяє передавати керування реактивним методам при появі аномальних навантажень.

Горизонтальні методи є більш масштабованими, оскільки не обмежені ресурсами однієї машини, проте керування відбувається лише на рівні кількості екземплярів. Запити на ОР на рівні екземпляру не корегуються в залежності від рівня утилізації кожного з ресурсів, що може призвести до неефективного використання частини ОР. Цю проблему можна вирішити використанням вертикальних методів, проте існуючі рішення не дозволяють комбінувати різні типи масштабування.

#### **3.1 Метод проактивно-реактивного масштабування**

Проактивне маштабування забезпечує високу ефективність використання ОР, а також дозволяє підтримувати необхідний рівень QoS. Проте проактивний тип масштабування є ефективним за умови наявності точних прогнозів, що можливо лише в типових ситуаціях. У випадку виникнення аномального навантаження, результати роботи алгоритмів прогнозування можуть призводити до порушень вимог до якості послуг та некоректного виділення ОР. З іншого боку, реактивний тип масштабування є значно менш ефективним [32, 33] при наявності точних прогнозів, проте може адаптуватися до нетипових навантажень, оскільки потребує лише даних про поточний стан ІТ-інфраструктури. Для покращення характеристик проактивного методу можливо інтегрувати реактивний компонент шляхом створення гібридного методу.

Гібридний метод, який включає проактивний і реактивний компоненти, є ефективним при прогнозованих навантаженнях внаслідок використання першої компоненти, так і при нетипових ситуаціях завдяки використанню другої. При розробці гібридних методів основним питанням є алгоритм визначення необхідності роботи проактивного чи реактивного компонентів. Окрім того, необхідно швидко визначати наявність нетипових навантажень задля мінімізації погіршення рівня QoS. Крім того, необхідно визначити умови, при яких використання проактивного методу є доречним.

### 3.1.1 Аналіз існуючих рішень для проактивно-реактивного масштабування

В роботі [64] автори презентують рішення CloudScale для управління ОР для мультитенантних хмарних систем. Дане рішення включає як проактивний, так і реактивний компоненти, які можуть працювати як разом, так і окремо в відповідних режимах роботи. В основу проактивного компонента покладено використання швидкого перетворення Фур'є для знаходження подібних шаблонів навантаження, після чого застосовуються ланцюги Маркова для ідентифікації поточного стану застосунку та потрібного переходу. Реактивний компонент в свою чергу обраховує помилку між поточним об'ємом зарезервованих ресурсів та необхідним, на основі чого об'єм виділених ресурсів коригується. Таким чином, реактивний компонент виконує коригувальну функцію. На різних наборах даних, розроблені компоненти в парі показали кращі результати ніж кожен з них окремо.

В роботі [65], автори пропонують інший варіант інтеграції реактивного та проактивного підходу, а саме, масштабування вгору відбувається реактивно, а вниз – проактивно. Реактивний компонент постійно аналізує поточні метрики швидкості відповіді на запити. У випадку, коли дані метрики порушуються SLA, відбувається масштабування вгору. Проактивний компонент, в основі якого використовується регресійна модель, запобігає передчасному звільненню зарезервованих ресурсів. Результати експериментів, що проводяться на синтетичних даних, демонструють здатність рішення до

адаптації до простих шаблонів навантаження. В роботах [66, 67] досліджуються можливі підходи для горизонтального масштабування. В результаті проведених експериментів в роботі приходять до висновку, що комбінація з двома незалежними контролерами – реактивного для масштабування вгору і проактивного для масштабування вниз – є найбільш ефективним з точки зору якості послуг. Проте навантаження розглядалося як таке, що не містить постійних сезонностей та трендів, а проактивне масштабування реалізовано на основі короткострокового прогнозування.

### 3.1.2 Сутність методу проактивно-реактивного масштабування

Основними компонентами при розробці гібридного проактивно-реактивного методу є безпосередньо реактивний і проактивний компоненти, які надають конфігурації ОР для застосунків та модуль прийняття рішення, який визначає доречність застосування першого чи другого типу масштабування. Загальна схема гібридного зображена на рис. 3.1:



Рисунок 3.1 – Загальна діаграма компонентів гібридного методу управління

При розробці гібридного методу необхідно виділити ситуації, в яких необхідно передавати керування між компонентами. При навчанні моделі прогнозування обов'язковим етапом є визначення її точності різними методами – розділення даних на навчальні та тестові, кросвалідація та інші [68]. Якщо отримана модель має низькі показники точності, то з високою

вірогідністю застосування прогнозів такої моделі може призвести до надлишкового або недостатнього резервування. Тому пропонується робити перехід з проактивного управління до реактивного при відсутності точних прогнозів. Така ситуація може виникнути через втрату даних або їх відсутність на початкових етапах роботи.

Можлива ситуація, коли точність отриманої моделі є достатньою для застосування прогнозів на інфраструктурі, але фактично прогнози не співпадають з актуальними потребами застосунків. Така ситуація може виникати через наявність нетипових навантажень, які відсутні в існуючих даних. В такому випадку, управління має переходити від проактивного до реактивного компонента, оскільки останній діє на основі лише поточного стану ІС. У випадку використання MAPE і поділу даних на тренувальні і тестувальні валідаційний індикатор описується як:

$$\frac{1}{n} \sum \left| \frac{X'_{ly}(t) - \hat{X}_{ly}(t)}{X_{ly}(t)} \right| < A_{threshold}, t = \overline{0, n}, l = \overline{1, m}, y = \overline{1, h}, \quad (3.1)$$

де  $X'_{ly}$  – прогнозоване значення використання  $l$ -го застосунку  $y$ -го ОР,  $\hat{X}_{ly}(t)$  – фактичне,  $A_{threshold}$  – констатне значення задовільного показника точності.

Якщо управління здійснюється реактивними методами, але при цьому була отримана достатня точність моделі, необхідно передати управління проактивному компоненту. Для запобігання повторенню попереднього випадку, в якому точність була достатня, але фактичні вимоги відрізнялися від прогнозованих, мають відбуватися додаткові перевірки.

Для вирішення цієї проблеми пропонується застосовувати асинхронний компонент моніторингу стану застосунків, який перевіряє наявність достатньої кількості ОР у застосунків або показники QoS перед прийняттям остаточного рішення щодо передачі управління від одного компонента до іншого. Такі ітерації відбувається з періодичністю  $T_{check}$ , а кількість неуспішних або успішних ітерацій для передачі управління визначається константами  $N_r$  та  $N_p$  відповідно.

$$\sum F(t_0 + T_{check} \cdot i) = N_r, i = \overline{1, N_r}, \quad (3.2)$$

де  $F(t)$  – індикаторна функція відповідності поточних запитів до поточних потреб, що повертає нуль при успішному результаті, а в інакшому випадку – один;  $t_0$  – початковий час відліку після першої невдачі. Таким чином, ця рівність виконується при послідовній невідповідності запитів до потреб ОР протягом заданої кількості ітерацій  $N_r$ , що є індикатором переходу від проактивного управління до реактивного.

Зворотня операція переходу описується наступним чином, де всі послідовні перевірки відповідності значень запитів ОР до фактичного використання мають бути успішними:

$$\sum F(t_0 + T_{check} \cdot i) = 0, i = \overline{1, N_p}, \quad (3.3)$$

де  $F(t)$  – індикаторна функція відповідності прогнозованих запитів до поточних потреб;  $t_0$  – початковий час відліку після отримання необхідної точності моделі прогнозування.

Перед безпосереднім застосуванням конфігурації на основі прогнозів здійснюється перевірка протягом вказаної кількості ітерацій, що підвищує стійкість ІС до повторних нетипових шаблонів навантажень. На рис. 3.2 зображена діаграма станів СУ та описані умови переходу між ними.



Рисунок 3.2 – Діаграма станів гібридного управління

Джерелом історичних даних для проактивного модуля виступає база даних системних метрик. В Kubernetes для цього використовується Prometheus. Після отримання даних і їх підготовки відбувається навчання

моделі та оцінка її точності методами MAPE, RMSE або іншими. Оцінка точності може відбуватися на відокремленій частині історичних даних. Така перевірка необхідна для уникнення ситуацій, коли короткостроково надається точний прогноз, але у довгостроковій перспективі точність значно падає. У разі достатньої точності, модуль прийняття рішень передає управління проактивному компоненту, який здійснює розрахунок необхідної кількості ОР на основі отриманих прогнозів. У інакшому випадку, управління передається реактивному компоненту.

Для визначення прийнятної точності необхідне встановлення константи  $A_{threshold}$ . Значення  $A_{threshold}$  має пряме відношення до підтримання необхідного рівня QoS. Низьке значення  $A_{threshold} < 80\%$  призведе до використання неточних прогнозів і некоректної конфігурації ОР, а занадто високе  $\sim 100\%$  до постійної роботи реактивного компонента, що має гірші характеристики ефективності з точки зору рівня QoS. Значення  $A_{threshold}$  може бути знайдено експериментально, оскільки точність алгоритмів прогнозування може відрізнятися на різних типах навантаження. На рис. 3.3 зображено алгоритм переходу від проактивного управління до реактивного.



Рисунок 3.3 – Алгоритм переходу між станами в залежності від отриманої точності прогнозів

У разі отримання точності моделі, що відповідає заданому  $A_{threshold}$ , але невідповідності прогнозованих значень фактичному використанню ресурсів модуль валідації здійснює постійні перевірки з метою виявити таку ситуацію.

Період перевірок  $T_{check}$  має відповідати періоду збору і оновлення метрик в ІС для швидкої реакції на зміну навантаження. Проте при низькому періоді ітерацій  $T_{check}$  кількість перевірок для переходу  $N$  має бути збільшена для того, щоб уникнути переходу від одного стану до іншого у випадку короточасних аномальних навантажень. На рис. 3.4 зображено схему алгоритму переходу від проактивного до реактивного управління.

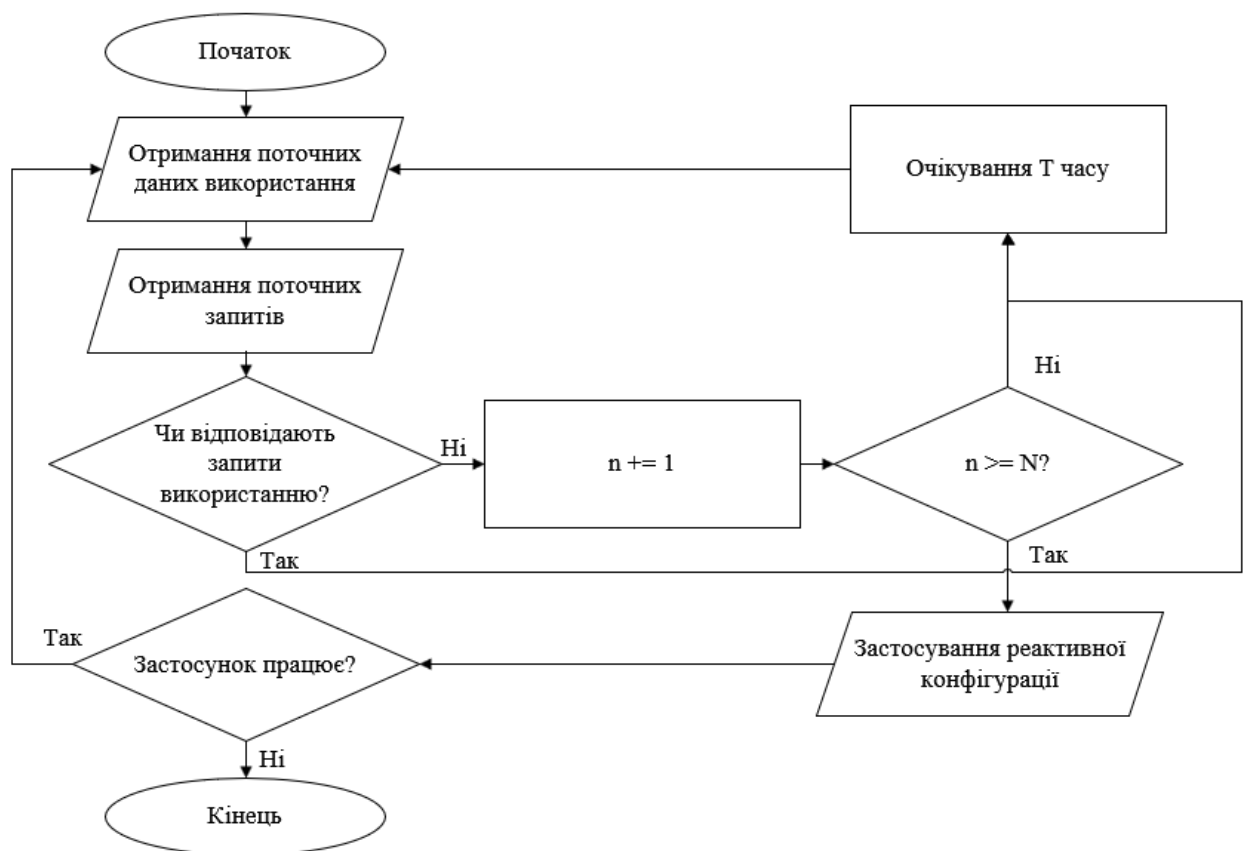


Рисунок 3.4 – Алгоритм переходу від проактивного до реактивного управління у випадку достатньої точності моделі

Для переходу до проактивного управління має виконуватися дві умови, а саме достатня точність отриманої моделі  $A_{model} > A_{threshold}$  і успішність перевірок протягом  $n$  ітерацій, що зображено на рис. 3.5. Це дозволить уникнути застосування моделі прогнозування низької точності та некоректних прогнозів навантаження.

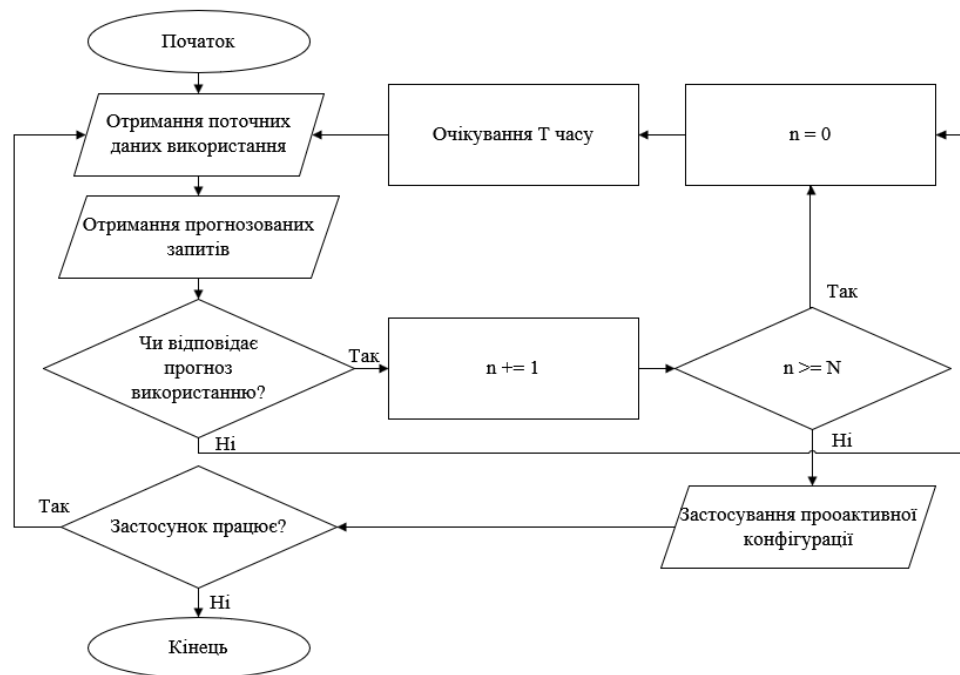


Рисунок 3.5 – Алгоритм переходу від реактивного до проактивного управління

В Kubernetes кластері реалізація гібридного методу вимагає наявності встановлених рішень для моніторингу, а саме Prometheus для роботи з високорівневими і агрегованими метриками та metrics-server для низькорівневих метрик поточного стану. Компонент metrics-server надає інформацію з меншими затримками, що доречно використовувати для реактивного масштабування. З іншого боку, проактивне масштабування вимагає наявності історичних даних, а Prometheus надає таку можливість з невеликими затримками. Модуль застосування призначений для роботи з Kubernetes API та безпосередньо відправляє команди на застосування нових конфігурацій OP.

Для проактивного масштабування використовується запропонований в розділі 2.4 метод. Для реактивного масштабування використовується HPA – вбудоване рішення для автоматизації горизонтального масштабування в Kubernetes. Основна ідея HPA полягає в тому, аби за допомогою регуляції кількості реплік підтримувати встановлене значення середнього навантаження на кожну репліку застосунку. Для мінімізації часу перемикання від проактивного до реактивного компоненту, HPA працює в режимі



спостереження, що досягається шляхом встановлення параметру `selectPolicy` в значення `disabled`.

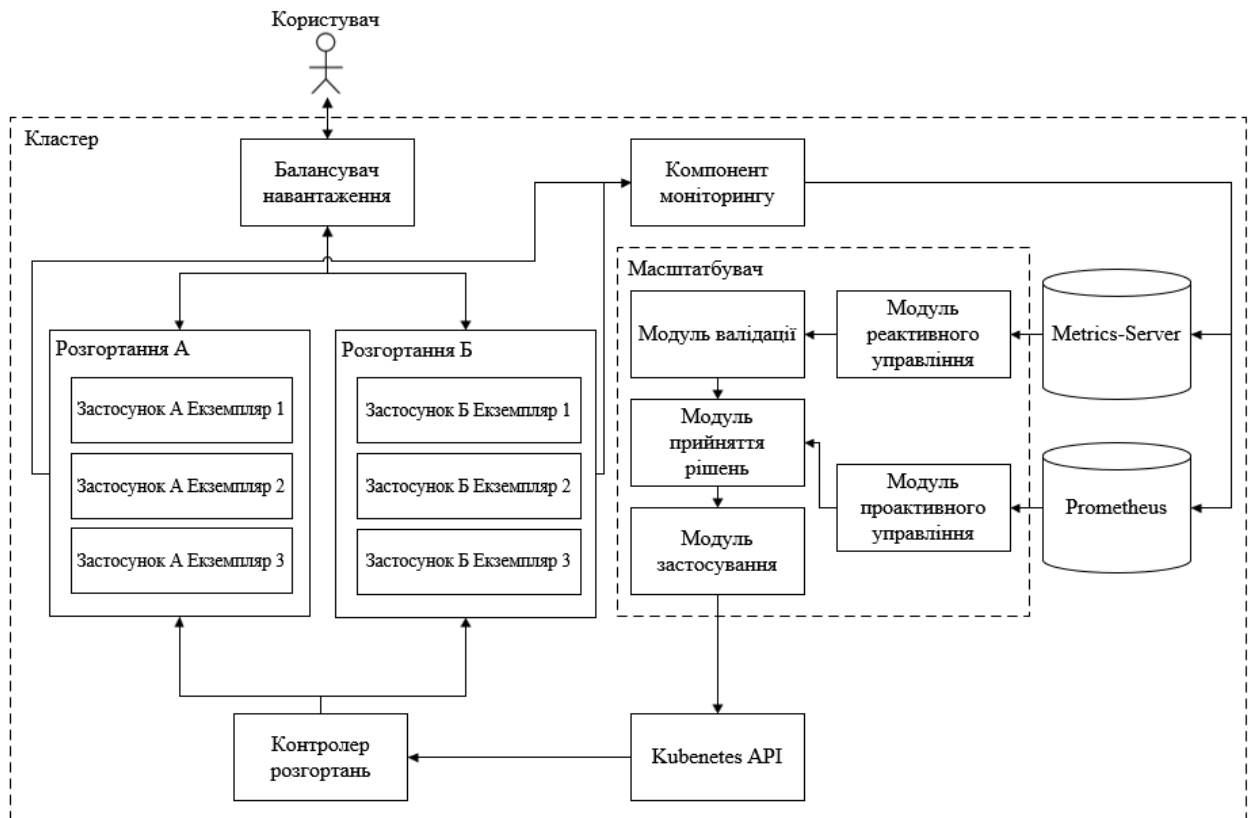


Рисунок 3.6 – Діаграма інтеграції гібридного рішення в Kubernetes кластер

### 3.1.3 Дослідження методу проактивно-реактивного масштабування

Для оцінки якості розробленого рішення проведено два експерименти на кластері `minikube`, який включає один вузол, має 12 процесорних ядер і 16 Гб оперативної пам'яті.

Навантаження генерується за допомогою утиліти `locust`, яка дозволяє надсилати запити по заданому сценарію. Тестовий застосунок може бути масштабований як горизонтально, так і вертикально. На кожний запит виконується синтетична задача, яка включає виконання операцій на CPU.

В першому експерименті розроблене рішення запускається без будь-якої попередньої історії. Періодичність навантаження становить п'ять хвилин та коливається від 20 до 100 запитів на секунду. На рис. 3.7 зображено результати проведеного експерименту. На початку управління ресурсами передається реактивному компоненту, через що в моменти появи навантаження присутні

високі значення часу відповіді. Після трьох періодів навантаження, проактивний компонент має змогу точно обчислювати необхідну кількість реплік та бере на себе управління. Починаючи з 800 секунди експеримента, час відповіді в момент появи навантаження значно зменшується, що відповідає часу переходу управління до проактивного компоненту.

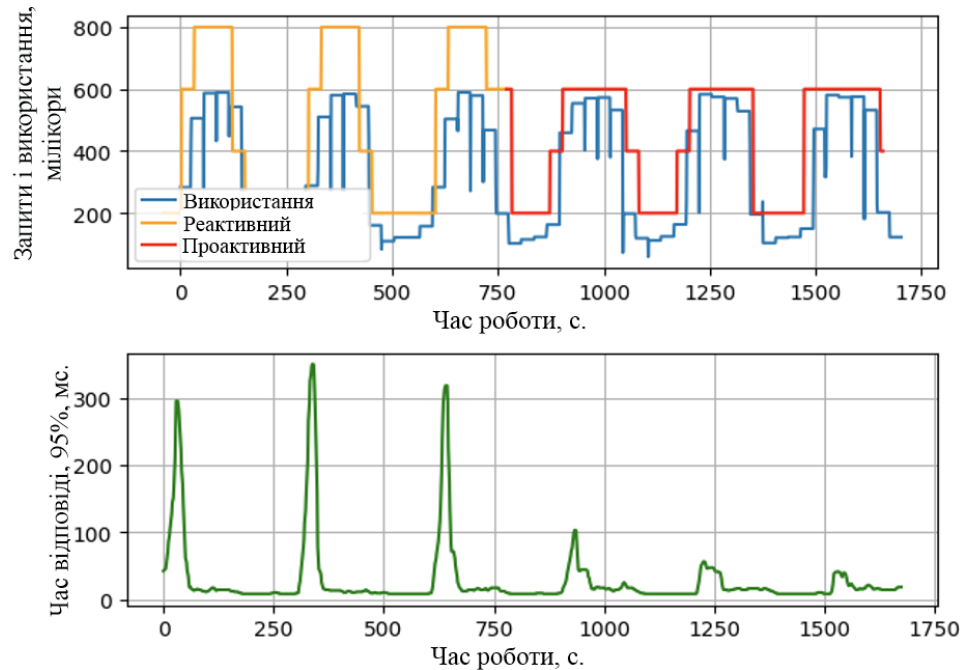


Рисунок 3.7 – Приклад переходу від реактивного до проактивного управління

Варто зазначити, що 95 перцентиль часу відповіді при проактивному масштабуванні на 210% кращий за аналогічний показник реактивного. Крім того, при проактивному управлінні резервувалося на 11% менше процесорного часу.

В другому експерименті визначається здатність зворотного переходу. Спочатку управління відбувається за допомогою реактивного компонента, але після появи нетипового навантаження управління переходить до реактивного компонента. Результати експерименту зображені на рис. 3.8. Протягом трьох періодів навантаження відповідає шаблону. Четвертий період містить аномальне навантаження, відповідно відбувається перехід до реактивного управління, оскільки виникає невідповідність запитів до фактичного використання ОР.

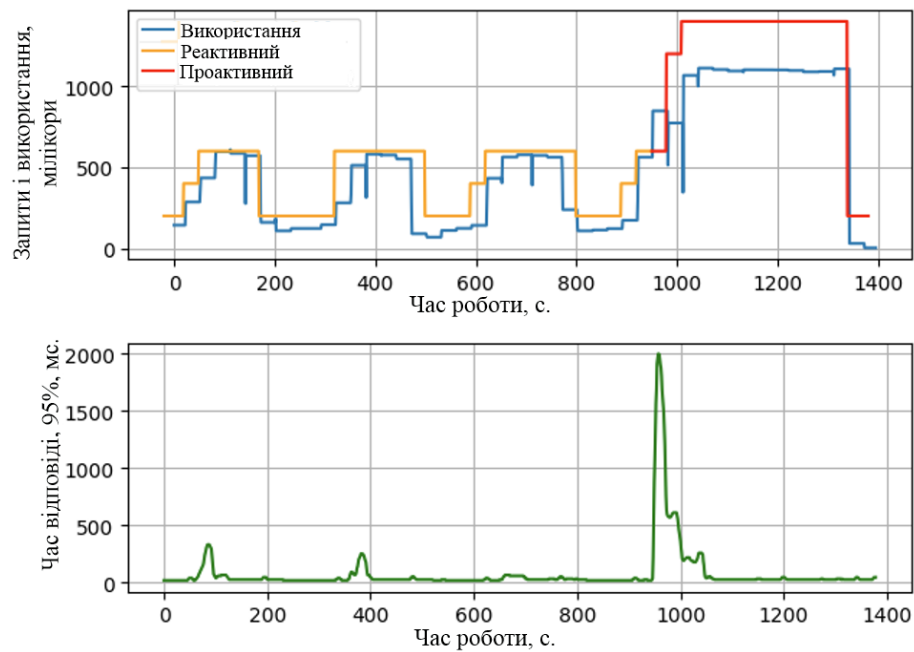


Рисунок 3.8 – Приклад переходу від проактивного до реактивного управління при нетиповому шаблоні навантаження

Проведені експерименти демонструють ефективність запропонованого методу проактивно-реактивного масштабування. Програмний модуль включає реалізації для горизонтального масштабування, проте запропонований метод може бути використаний і для вертикального управління з використанням аналогічних індикаторів переходу.

### 3.2 Метод горизонтально-вертикального масштабування

Горизонтальне та вертикальне масштабування застосунків є одним із найбільш ефективних інструментів для автоматизації процесів управління ОР. Горизонтальне масштабування, на відміну від вертикального, є більш універсальним, оскільки забезпечує відмовостійкість, не є обмеженим ресурсами однієї фізичної машини та дозволяє гнучко керувати об'ємами виділених ресурсів [69]. На практиці вертикальне масштабування використовується у випадку, коли застосунок не може бути розподілений. Хоча горизонтальне масштабування є гнучким способом динамічно розподіляти ресурси, цей процес відбувається в межах встановлених запитів. Якщо кількість виділених застосунку ОР і актуальне навантаження по

кожному з ресурсів не збалансоване, то відповідно частина цього ресурсу не використовується. Регулювання об'ємів запитів в межах одного застосунку може дозволити розміщення його екземплярів на більш спеціалізованих ВМ, що дозволить знизити фінансові витрати при еквівалентному рівні QoS [70].

У першому розділі запропонована модель горизонтального масштабування. Описана модель горизонтального масштабування має недолік – недостатня утилізація неперіоритетних типів ресурсів. Неперіоритетний ресурс не має впливу на об'єми виділення ресурсів під планування в цілому або в конкретний момент часу. На рис. 3.9 зображено приклад незбалансованого виділення ресурсів для застосунку, коли утилізація процесорного часу підтримується на заданому рівні, що становить 90%, але рівень утилізації інших ресурсів є низьким.

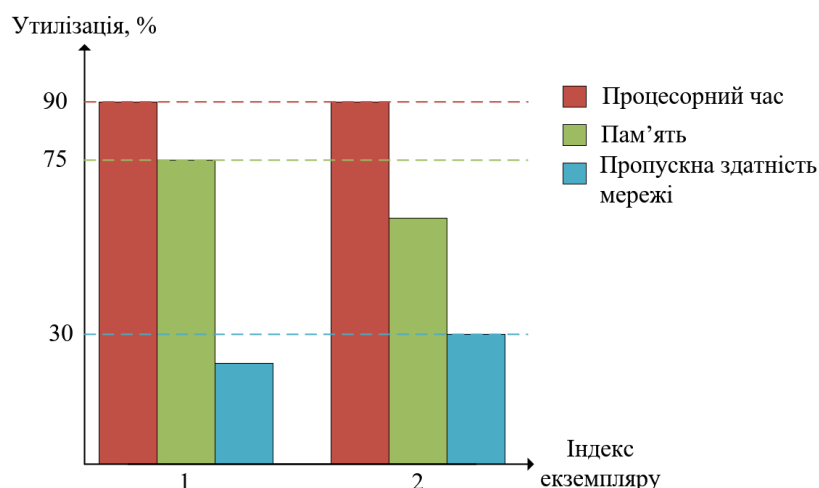


Рисунок 3.9 – Незбалансована утилізації ОР при горизонтальному масштабуванні

Зменшення кількості екземплярів неможливе через необхідність забезпечувати заданий рівень QoS, а часткове регулювання об'єму ресурсів застосунку не може бути виконане компонентом горизонтального масштабування. Для вирішення описаної проблеми необхідно в ручному режимі адаптувати конфігурацію запитів та горизонтального масштабування до поточних потреб, що є складною задачею в великих ІС.

Пропонується метод гібридного масштабування для автоматизації даного процесу, який є похідним від горизонтального, але при цьому є більш ефективним в контексті утилізації ОР. Запропонований метод покликаний вирішити наступні проблеми при роботі застосунків, що масштабуються горизонтально:

- підвищення рівня утилізації типів ресурсів, які не є пріоритетними при горизонтальному масштабуванні, для підвищення ефективності їх використання;

- урегулювання ситуацій, коли максимальне використання екземпляром одного з типів ресурсів перевищує встановлені запити на цей тип ресурсу, зокрема, помилки нестачі пам'яті, які призводять до відмови від обслуговування.

### 3.2.1 Аналіз існуючих методів інтеграції вертикального і горизонтального масштабування

Тема комбінації вертикального і горизонтального масштабування все частіше з'являється в наукових працях [40]. Більшість підходів базується на прогнозуванні. В роботі [71] пропонується підхід з використанням управління прогнозуючими моделями з використанням моделі із заданою кількістю параметрів, які визначаються на основі ємності кластера, прогнозованого об'єму черги задач і запитів застосунків на ОР. На основі заданих обмежень і отриманих прогнозів вирішується задача оптимізації розподілення ОР. Експериментально отримані метрики QoS комбінованого методу показують приріст у порівнянні з горизонтальним масштабуванням, проте оцінка ефективності використання ОР не вивчається. В роботі [72] пропонується інший підхід, коли ВМ масштабується вертикально, поки не досягнуто межі фізичної машини, після чого застосовується горизонтальне масштабування.

В даній роботі припускається, що горизонтальне масштабування може забезпечувати необхідний рівень QoS, тому значна увага приділяється саме ефективності використання ОР кластера. Також, метод, що пропонується, здатний використовувати як історичні дані, так і дані методів прогнозування.

### 3.2.2 Розробка координатора

Для успішної спільної роботи горизонтального і вертикального масштабування необхідна координація їх роботи. Координування має здійснюватися таким чином, що зміни внесені кожним компонентом є повністю узгодженими [73]. Необроблені конфліктуючі зміни можуть призвести до падіння рівня QoS та відмови від обслуговування. Модуль координації містить дані про особливості роботи і можливості кожного з методів масштабування.

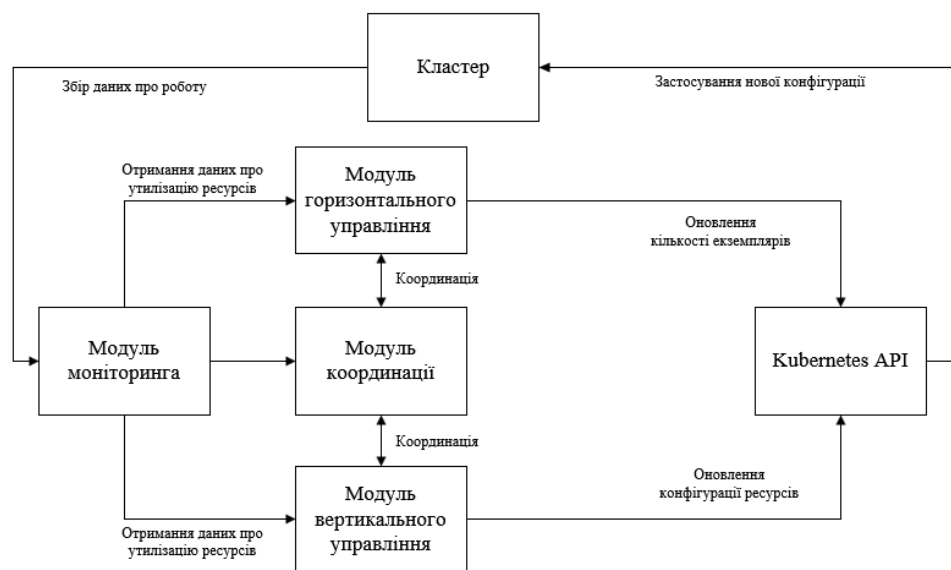


Рисунок 3.10 – Діаграма компонентів гібридного методу масштабування

В модулі координації для узгодження роботи компонентів пропонується використовувати принцип пріоритетності, коли один з компонентів є другорядним і не може впливати на поточні рішення компонента управління з вищим пріоритетом, а лише адаптуватися до наданих рішень. Нескоординоване прийняття рішень може призвести до некоректного масштабування застосунку і до падіння показників QoS. Модуль координації забезпечує узгодження конфігурацій компонентів масштабування наступним чином:

$$S(x) = \begin{cases} H(x, V(x, null)), P_v > P_h, \\ V(x, H(x, null)), P_h > P_v, \end{cases} \quad (3.4)$$

де  $S(x)$  є функцією масштабування,  $H(x, c)$  та  $V(x, c)$  – функції масштабування горизонтального та вертикального компонентів відповідно, які приймають на вхід історичні дані та наявну конфігурацію,  $P_H$  та  $P_V$  є пріоритетами відповідних компонентів [69].

Модуль координації надає перевагу саме горизонтальному масштабуванню, оскільки такий підхід не обмежений ресурсами однієї фізичної машини та є більш гнучким для використання в сучасній архітектурі з використанням мікросервісів.



Рисунок 3.11 – Порівняння характеристик типів масштабувань

Для реалізації запропонованого методу можна використовувати як джерела історичних даних утилізації застосунків, так і дані отримані в результаті роботи моделей прогнозування. Історичні дані передаються до модуля горизонтального управління, де приймається рішення про доцільність збільшення або зменшення кількості екземплярів. На вхід модуля горизонтального управління також подається інформація про поточні запити на ресурси  $X_{ly}$ ,  $l=1, \dots, m$ ,  $y=1, \dots, h$ , де  $m$  – кількість типів ресурсів для управління,  $h$  – кількість застосунків і цільовий рівень утилізації кожного з ресурсів  $T_{ly}$ ,  $0 < T_{ly} \leq 1$ . Результатом роботи модуля горизонтального управління є кількість екземплярів  $l$ -го застосунку  $k_l$ . Прикладом такого модуля є НРА.

В модулі вертикального масштабування на першому етапі відбувається визначення рівня утилізації кожного з ОР:

$$U_{ly} = \frac{X_{ly}}{\hat{X}_{ly}}, l = 1, \dots, m, y = 1, \dots, h, \quad (3.5)$$

де  $U_{ly}$  – рівень утилізації  $l$ -го застосунку  $y$ -го типу ОР.

На основі встановлених цільових рівнів утилізації  $T_{ly}$  визначається опорний тип ресурсу, на основі якого відбувалося горизонтальне масштабування:

$$y_{pivot} = \arg \max \left( \frac{U_{ly}}{T_{ly}} \right), l = 1, \dots, m, y = 1, \dots, h, \quad (3.6)$$

де  $y_{pivot}$  – індекс ресурсу з максимальним відносним рівнем утилізації  $l$ -го застосунку  $y$ -кого типу ОР. Наступним кроком відбувається оцінка необхідних об'ємів ресурсів, які не є опорними з використанням цільових рівнів утилізації  $T_{ly}$ :

$$\begin{cases} \hat{X}_{ly} = f(\{X_{ly}(t_0), X_{ly}(t_1), \dots, X_{ly}(t_n)\}, T_{ly}), l = 1 \dots m, y = 1 \dots h, n = 1 \dots N, \\ y \neq y_{pivot}, \end{cases} \quad (3.7)$$

де  $\hat{X}_{ly}$  – обчислені на основі історичних даних або прогнозів значення запитів на ОР на поточній ітерації, для  $X_{ly}(t)$  відбувається перехід від скалярного значення до функції, оскільки для обчислення запитів необхідно враховувати відрізок історичних даних або отриманих прогнозів;  $f(x, y)$  – функція обчислення запитів. В залежності від конфігурації, для цієї функції можуть застосовуватися різні методи оптимізації.

При масштабуванні по всім доступним ОР пропонується застосовувати універсальну функцію, результат якої гарантує коректну роботу застосунків та компонентів масштабування:

$$f(x, u) = \frac{\max(x)}{u}, \quad (3.8)$$

де  $x$  – вектор значень використання ОР, а  $u$  – цільове значення його утилізації.

Якщо  $y_{pivot}$  є фіксованим значенням, то можуть застосовуватися наступні оптимізації.



Кожен тип ОР має свою специфіку управління. Пропонується розділити їх на дві групи. Перша група включає типи ресурсів, перевищення використання яких не призводить по повної відмови від обслуговування завдяки амортизації нестачі на наступних циклах роботи застосунку [74]. Такі типи ОР можуть бути штучно обмежені з метою дотримання встановлених вимог. Наприклад, для CPU ресурсу застосунку буде надаватися лише встановлений час роботи на ядрі процесора, а всі операції, які виходять за межі, будуть виконані в наступному періоді роботи [75]. Схожий принцип роботи має мережевий ресурс, в якій створюється черга з пакетів на відправку. Для ОР з першої групи пропонується використовувати заданий перцентиль при обчисленні запитів:

$$f_1(x, p) = x_{sorted} \left[ \left\lceil \frac{p}{100} \cdot (n-1) \right\rceil \right], \quad (3.9)$$

де  $x$  – вектор значень використання деякого ОР,  $n$  – довжина вектору  $x$ , а  $p$  – цільовий перцентиль утилізації. Запропонована оптимізація може бути корисною при наявності короткочасних пікових значень у навантаженні застосунку. Перцентиль дозволяє обчислити такі запити, які покриватимуть більшість наявних сценаріїв навантаження та дозволять уникнути надмірного резервування ресурсів.

### 3.2.3 Експериментальні дослідження горизонтально-вертикального методу

Для оцінки ефективності запропонованого методу проводиться моделювання роботи застосунку, який масштабується як горизонтально, так і вертикально. Проведення експериментів на Kubernetes інфраструктурі ускладнюється тим фактом, що для оновлення запитів для застосунку необхідно перезапустити всі його екземпляри. На реальній інфраструктурі це досягається шляхом поступового перезапуску кожного з екземплярів, що не вимагає суттєвих додаткових об'ємів ОР та не впливає на метрики QoS. Моделювання відбувається засобами Python та бібліотеки Pandas. Модель

горизонтального масштабування аналогічна до НРА, а саме використовується реактивний підхід, визначаються цільові рівні утилізації для процесорного часу та пам'яті, а переоцінка запитів відбувається з періодичністю в 100 с, що є відображенням періодичності збору та агрегації метрик в реальному кластері.

Перший експеримент перевіряє роботу запропонованого методу, коли опорним ресурсом, по якому відбувається горизонтальне масштабування є процесорний час. Відповідно вертикально масштабується пам'ять. В першому дослідженні сумарне навантаження з періодичністю 1000 с коливається від 200 мілікорів до 1200 мілікорів, а встановлені запити на процесорний час становлять 250 мілікорів при цільовому рівні утилізації 90%. Таким чином застосунок масштабується від двох до п'яти екземплярів. На рис. 3.11 зображено графік навантаження та кількості екземплярів для його обробки. Також на рис. 3.12 можна побачити затримку масштабування при появі навантаження і відповідно затримку перед зменшенням кількості екземплярів, що відповідає поведінці НРА.

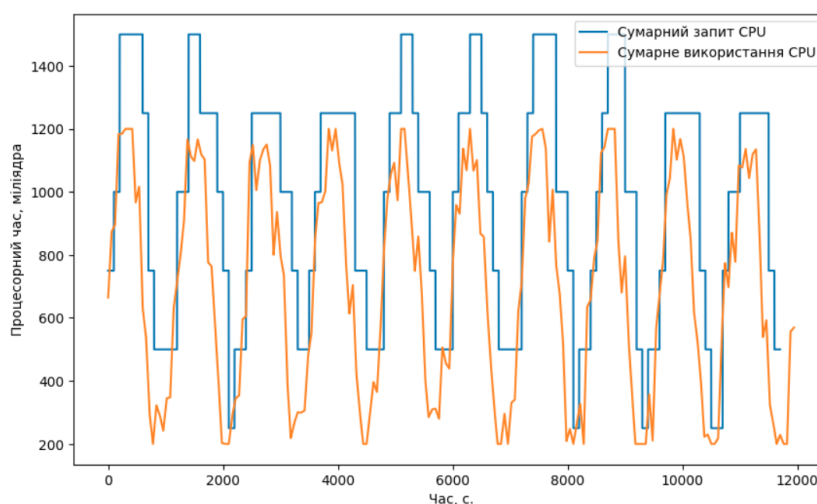


Рисунок 3.12 – Горизонтальне масштабування застосунку по процесорному часу

Початкова конфігурація запитів пам'яті не є оптимальною та становить 100 Мб. При цьому в моделі враховується, що кожен екземпляр має постійну складову використання пам'яті та таку, що залежить від навантаження.

Відповідно на перших двох періодах роботи утилізація пам'яті становить 50%. Після того, як компонент отримав достатню кількість даних, а саме протягом 2000 с, відбувається масштабування вниз для запитів на пам'ять до цільового рівня 90%. На четвертому періоді роботи з'являється аномальний пік використання пам'яті і компонент вертикального масштабування відповідно реагує збільшенням запитів на пам'ять. Після двох періодів низького рівня утилізації знову відбувається масштабування пам'яті вниз до рівня утилізації 90% протягом двох останніх періодів роботи. Результати дослідження зображені на рис. 3.13.

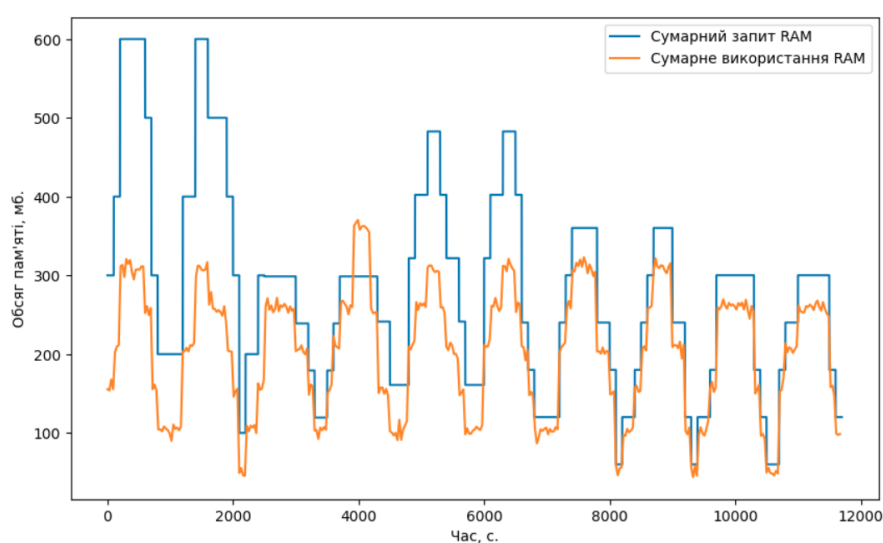


Рисунок 3.13 – Вертикальне масштабування запитів пам'яті

Наступний експеримент націлений на оцінку роботи методу, коли горизонтальне масштабування відбувається по пам'яті, а вертикальне масштабується процесорний час. На рис. 3.14 зображене навантаження на застосунок, а також кількість екземплярів застосунку. Різниця у порівнянні з першим експериментом полягає у використанні типу ресурсу, перевищення використання якого не призводить до аварійного завершення застосунку. Відповідно до рівняння (3.9) розрахунок оптимального значення запитів для процесорного часу відбувається на основі вказаного перцентилля, що дозволяє ефективно утилізувати ОР.

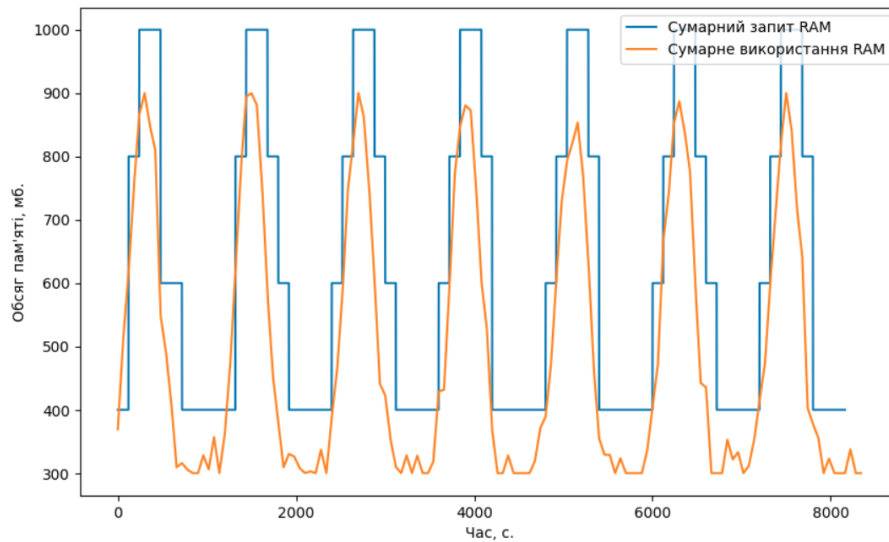


Рисунок 3.14 – Горизонтальне масштабування застосунку по пам'яті

На рис. 3.15 відображені результати вертикально масштабування. Для обрахунку запитів на процесорний час використовується 95 перцентиль. На рис. 3.15 можна побачити аналогічну роботу по масштабуванню вниз та вгору і відповідних ситуаціях. Різниця з першим експериментом полягає в вищому рівні утилізації ресурсу, що масштабується вертикально.

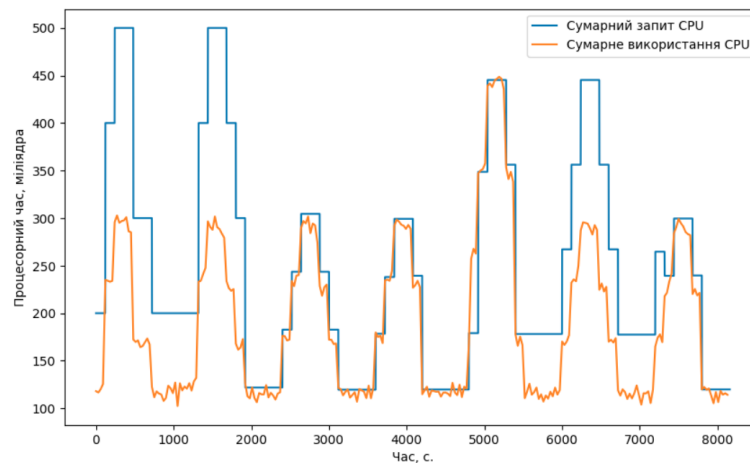


Рисунок 3.15 – Вертикальне масштабування запитів на процесорний час

Останній експеримент націлений на оцінку ефективності в залежності від довжини історичних даних, що враховуються при вертикальному масштабуванні. Горизонтальне масштабування відбувається по пам'яті з періодичністю 1000 с, що зображено на рис. 3.16.

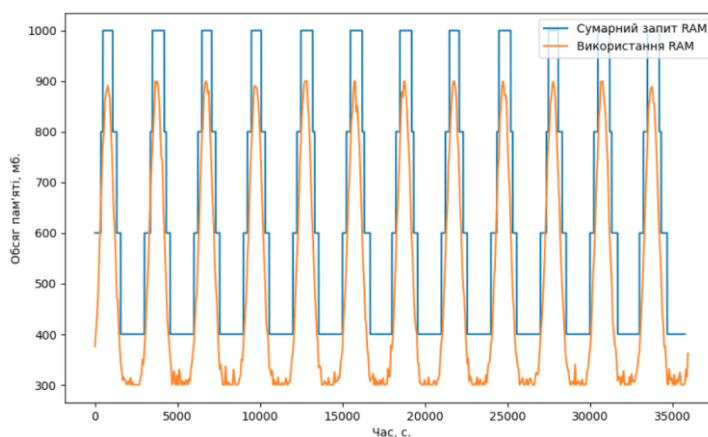


Рисунок 3.16 – Горизонтальне масштабування застосунку по пам'яті

На рис. 3.17 зображено результати роботи компонента вертикального масштабування, а розмір історичних даних, що беруться до уваги, становить 1000 с. Цільова утилізація процесорного часу становить 95%.

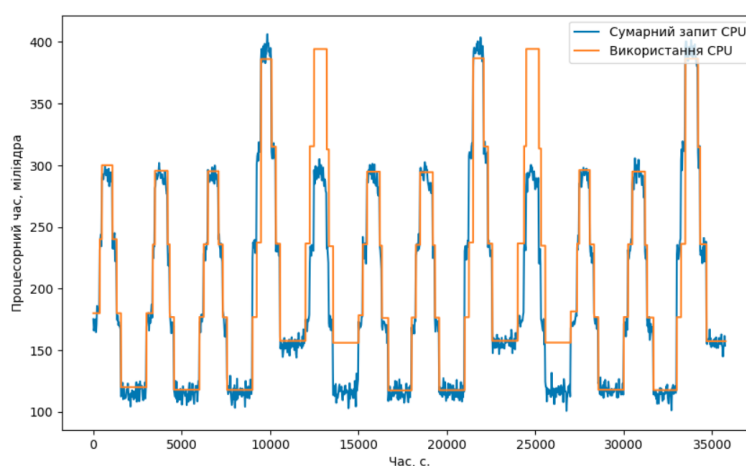


Рисунок 3.17 – Вертикальне масштабування застосунку по процесорному часу з довжиною історичних даних 1000 с

Аналогічний експеримент зображений на рис. 3.18, але розмір історичних даних збільшений до 4000 с. Збільшення довжини історичних даних, що використовуються для обчислення оптимальної конфігурації ОР дозволить враховувати довші шаблони і сезонності навантаження, що необхідним для забезпечення необхідного рівня якості послуг.

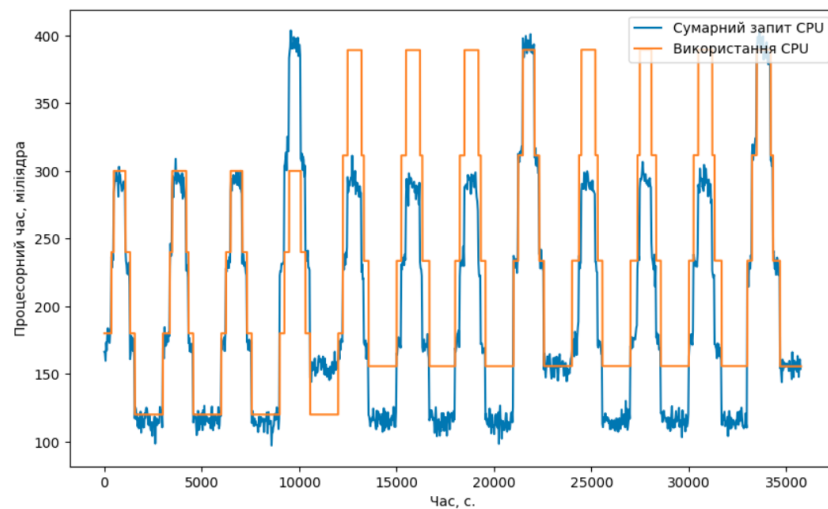


Рисунок 3.18 – Вертикальне масштабування застосунку по процесорному часу з довжиною історичних даних 4000 с

Результати показують, що масштабування відбувається зі значно меншим періодом, що з одного боку призводить до меншої ефективності використання ресурсів, а з іншого до кращих показників QoS при пікових навантаженнях. Проведений експеримент демонструє можливість адаптувати розроблений метод під встановлені вимоги SLA.

Далі проводиться порівняння утилізації процесорного часу при статичних і динамічних запитах, що зображено на рис. 3.19. Утилізація при статичних запитах в середньому становить 60%. При використанні динамічних запитів середня утилізація становить 90%.

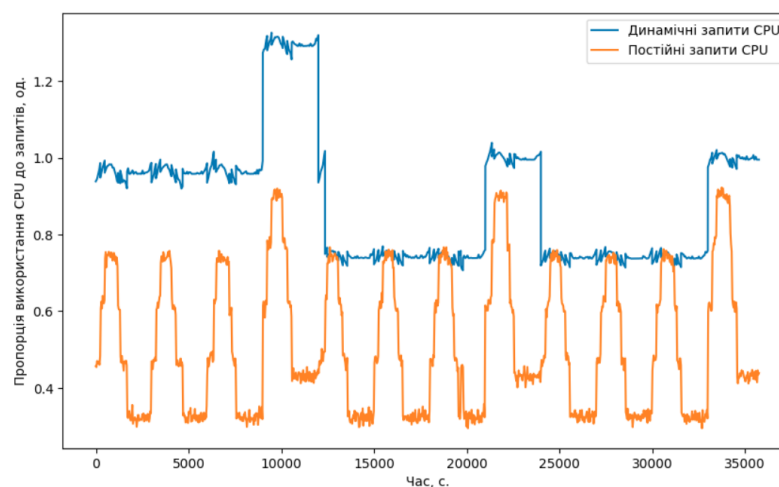


Рисунок 3.19 – Порівняння утилізації ресурсів при статичному і динамічному управлінні

Рівень QoS в даному випадку залежить від багатьох інших факторів, зокрема рівень встановлених лімітів. При цьому використання статичних запитів, на рівні 95 перцентилля, призводить до низького рівня утилізації від 30% до 90% в наведених умовах.

Обсяг виділеного процесорного часу при динамічних запитах у порівнянні зі статичними є меншим на 65%. Проведені експерименти є моделюванням і на реальній інфраструктурі результати можуть значно відрізнятись, проте вони дозволяють оцінити теоретичну можливість застосування розробленого методу.

### **Висновки до розділу 3**

В розділі доведено необхідність використання гібридних методів масштабування. Розглянуті існуючі комбіновані методи масштабування і методи комбінації для гібридних рішень.

Запропоновано комбінований метод, який включає реактивний і проактивний компонент для покращення стійкості проактивного підходу до аномальних навантажень і роботи в умовах повної або часткової відсутності даних. Запропонований метод перевірено експериментальним шляхом і продемонстровано ефективність даної комбінації.

Проаналізовано недоліки горизонтального масштабування як для проактивного, так і реактивного підходів. На основі аналізу запропоновано комбінацію вертикального і горизонтального масштабування. Експериментально доведено, що задля узгодження роботи компонентів доцільно використовувати координатор. Для оцінки ефективності запропонованого методу проведено симуляцію роботи застосунку. Результати експериментів демонструються зменшення резервування об'єму ОР на 65% в моделюванні та підтверджують доцільність використання гібридних методів масштабування.

## 4 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ УПРАВЛІННЯ ОБЧИСЛЮВАЛЬНИМИ РЕСУРСАМИ

В даному розділі на основі запропонованих моделей і методів розробляється інформаційна технологія управління ОР. Вирішуються основні задачі, обираються компоненти, розробляється структура та визначаються особливості інтеграції в існуючі рішення керування ІТ-інфраструктурою. В роботі [1] запропонована інформаційна технологія для здійснення управління інфраструктурою з дотриманням встановленого рівня якості послуг при мінімізації використання наявних ресурсів, проте не розглядається управління ОР в контексті проактивного масштабування та в умовах наявності великої кількості різнотипних застосунків в межах однієї ІС.

Інформаційна технологія управління ОР є сукупністю методів, що забезпечують ефективне використання ОР у віртуальних або фізичних кластерах. Для впровадження інформаційної технології в систему управління треба визначити методи управління, а також обрати технічні засоби та створити ПЗ. Сукупність цих складових забезпечує виконання інформаційних процесів задля підвищення надійності надання інформаційних послуг на фоні ефективного використання задіяних ресурсів [1].

### 4.1 Особливості обчислювальними ресурсами і навантаженням у кластері

В підрозділі розглядаються аспекти управління ОР, які безпосередньо не пов'язані з розподіленням ОР між застосунками у кластері. Визначення коректних політик перевищення запитів ОР, порядку розміщення екземплярів застосунків та балансування навантаження є необхідними аспектами для коректного функціонування ІС та виконання (1.1).

4.1.1 Політики управління при перевищенні запитів на обчислювальні ресурси

Згідно з (1.2) сумарні запити застосунків на ОР  $X_{lyi}$ ,  $l = \overline{1, m}$ ,  $y = \overline{1, h}$ ,  $i = \overline{1, n}$ , де  $h$  – кількість типів ресурсів для управління,  $m$  – кількість застосунків,  $n$  –



кількість вузлів, не можуть перевищувати загальний об'єм ОР  $i$ -го вузла  $V_{iy}$ , що гарантує наявність необхідної кількості ОР для роботи всіх розміщених застосунків. Необхідно визначити алгоритм роботи при перевищенні фактичного використання ресурсів  $\hat{X}_{ly}$  встановлених вимог. Деякі типи ресурсів можна обмежити без повної зупинки екземпляра, зокрема, може застосуватися тротлінг у випадку процесорного часу та пропускну здатності мережі. При перевищенні використання заданих об'ємів ресурсів екземпляром застосунка в роботі пропонується використання двох політик обробки такої ситуації – строгої та нестрокої.

При строгій політиці у випадку перевищення  $\hat{X}_{ly}$  встановлених вимог  $X_{lyi}$ ,  $l = \overline{1, m}$ ,  $y = \overline{1, h}$ ,  $i = \overline{1, n}$ , де  $h$  – кількість типів ресурсів для управління,  $m$  – кількість застосунків,  $n$  – кількість вузлів, відбувається негайне завершення екземпляру, що порушує встановлені запити ОР. У такий спосіб гарантується, що екземпляри застосунків не впливатимуть на роботу інших суміжних екземплярів на вузлі розміщення. Цю запропоновану політику можна застосувати для менш важливих застосунків.

При нестрогій політиці розміщенні екземпляри можуть використовувати надлишкові ресурси вузла доти, поки сумарне фактичне використання не перевищує потужність даного вузла:

$$\Delta X_{lyi} = V_{yi} - \sum X_{lyi} \leq, l = \overline{1, m}, y = \overline{1, h}, i = \overline{1, n}, \quad (4.1)$$

де  $\Delta X_{lyi}$  – об'єм доступних ресурсів вузла для використання за межами встановлених запитів,  $X_{lyi}$  – вимоги екземпляра  $l$ -го застосунку до ОР для  $y$ -го типу ресурсу на  $i$ -му вузлі кластера;  $V_{yi}$  – об'єм ОР доступний для  $i$ -го вузла для  $y$ -го типу ресурсу.

При цьому в разі планування розгортання нових екземплярів, використовуються саме встановлені запити, а не фактичне розгортання. Прикладом реалізації таких політик є механізм requests і limits та механізм пріоритетів в Kubernetes.

#### 4.1.2 Масштабування кластера

Задача щільного розміщення екземплярів застосунків у кластері полягає в оптимізації розгортання робочих навантажень на доступних вузлах для максимізації рівня утилізації [76]. Вирішення цієї задачі дозволить зменшити фінансові витрати та пришвидшити процес розгортання нових екземплярів. Необхідно зазначити, що ця задача є підмножиною задач оптимального розміщення віртуальних машин, яка є NP-повною [77].

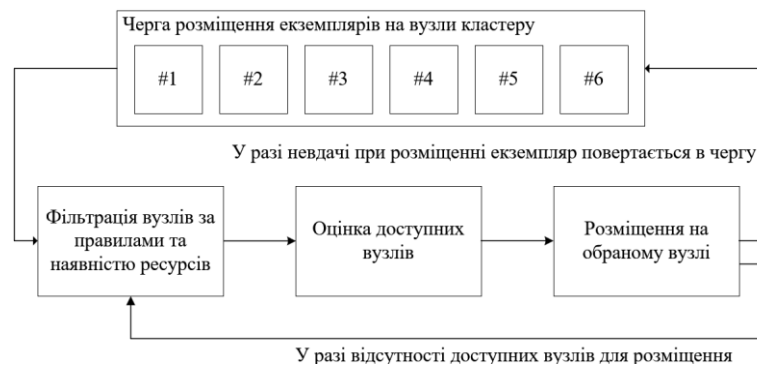


Рисунок 4.1 – Схема етапів розміщення екземплярів

У випадку, якщо кластер не має достатньої кількості ОР для розгортання нового екземпляру, має відбуватися масштабування кластера шляхом додавання нових вузлів таким чином, щоб виконувалася нерівність (1.3) з урахуванням екземплярів в черзі на розгортання. Повна утилізація ОР кластера означає, що розміщення нових екземплярів чи застосунків або неможливе, або відбувається зі значною затримкою. Для покращення метрики часу розгортання введемо коефіцієнт запасу.

$$\begin{cases} \sum V_{yi} \geq (1 + \varepsilon) \cdot \sum X_{ly} \\ \frac{\sum X_{ly}}{\sum V_{yi}} \rightarrow \max \end{cases}, l = \overline{1, m}, y = \overline{1, h}, i = \overline{1, n}, \quad (4.2)$$

де  $X_{ly}$  – вимоги екземпляра  $l$ -го застосунку до ОР для  $y$ -того типу ресурсу, що розміщується у кластері;  $V_{ly}$  – об'єм ОР доступний для  $i$ -того вузла для  $y$ -того типу ресурсу;  $\varepsilon$  – коефіцієнт запасу, який визначається наступним чином:

$$\begin{cases} \varepsilon = \frac{\sum V'_{iy}}{\sum V_{iy}} - 1, y = \overline{1, h}, i = \overline{1, n}, \\ V'_{iy} \geq V_{iy} \end{cases} \quad (4.3)$$

де  $\sum V'_{iy}$  – максимальна очікувана потужність кластера в майбутньому, а система зводиться до наступної залежності:

$$\frac{\sum X_{ly}}{\sum V_{yi}} \rightarrow \frac{1}{1 + \varepsilon}, l = \overline{1, m}, y = \overline{1, h}, i = \overline{1, n}. \quad (4.4)$$

При  $\varepsilon > 1$  кластер має достатньо ОР для швидкого розгортання нових застосунків або екземплярів в рамках встановлених обмежень SLA і при цьому є оптимальним з фінансової точки зору. Зокрема, якщо розгортання нових застосунків або масштабування існуючих не передбачається, тоді при  $\varepsilon = 1$  утилізація ресурсів кластера має становити 100%.

#### 4.1.3 Балансування навантаження між екземплярами застосунків

Однією з вимог до інформаційної технології управління контейнеризованими застосунками є здатність коректно балансувати вхідні запити або задачі на обробку між екземплярами застосунків.

Балансування – це процес розподілення робочого навантаження між множиною екземплярів застосунку з метою зниження затримок, підвищення пропускної здатності та забезпечення відмовостійкості, що напряду впливає на рівень QoS і дотримання встановлених вимог SLA [78].

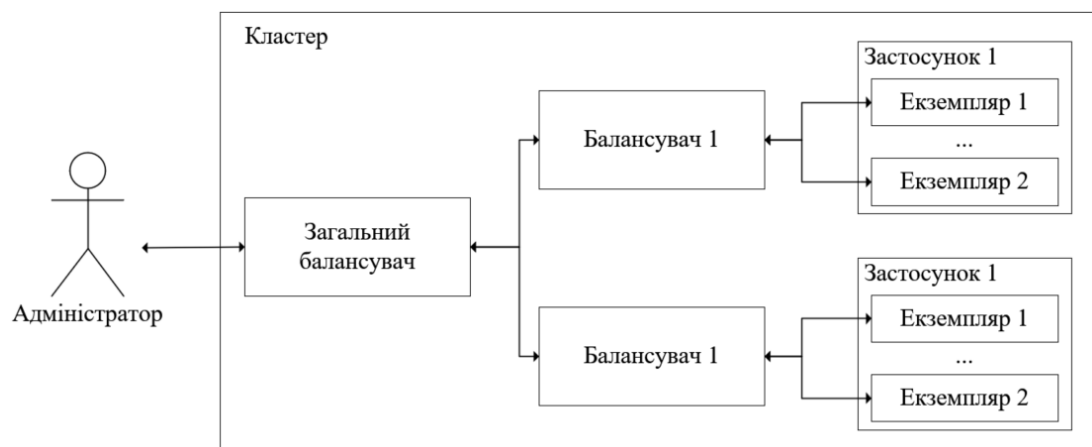


Рисунок 4.2 – Схема балансування у кластері

Балансування навантаження відбувається на спеціалізованому програмному сервері, який розташований безпосередньо перед екземплярами застосунків. Крім балансування, може також надаватися функціонал логічного розподілу, наприклад, направлення запитів одного користувача на один і той самий екземпляр з метою використання локального кеша екземпляра.

На рис. 4.2 зображено декілька балансувачів. Загальний балансувач виконує функцію логічного розподілу і переадресовує запити на балансувач застосунку згідно встановлених правил за допомогою заголовків або інших атрибутів запитів. Балансувач застосунку при розподіленні запитів між екземплярами використовує встановлені політики.

Підвищення пропускної здатності досягається шляхом рівномірного розподілення запитів так, що кожен екземпляр повністю утилізує надані ОР вузла і не відбувається перевантаження жодного з них.

Відмовостійкість забезпечується шляхом переадресації запитів з екземпляра, який не може виконувати свої функції, на інші екземпляри застосунку при наявності запасу пропускної здатності.

Некоректний розподіл запитів може призвести до перевантаження конкретного екземпляру, що може мати ряд наслідків. По-перше, це зупинка роботи екземпляра через вихід за межі запитів на ОР, що призводить до додаткового навантаження на інші екземпляри і ланцюгового ефекту, що може призвести до повної відмови від обслуговування. Іншим наслідком є нерівномірність використання ОР при моніторингу та аналізу застосунків. Дані про споживання ОР можуть використовуватися для подальшої їх оптимізації, що значно ускладнюється наявністю нерівномірного розподілення запитів.

Методи балансування можуть бути статичними і динамічними [79, 80]. Статичні методи є ефективними в середовищах з передбачуваним навантаженням, де варіативність запитів є мінімальною. Статичні методи не вимагають адаптації конфігурації та використовують задані правила для розподілення запитів між екземплярами. Основною перевагою статичних

методів є простота, детермінованість та низькі вимоги до моніторингу стану екземплярів. Прикладом таких методів є Round-Robin [79], де екземпляр до обробки обирається наступним чином:

$$l(x) = (l(x-1) + 1) \bmod k, \quad (4.5)$$

де  $l(x)$  – функція для вибору індексу екземпляру,  $l(x-1)$  – значення функції на попередньому кроці, а  $k$  – кількість екземплярів. Таким чином кожен з екземплярів по чергово отримують запити.

Динамічні методи балансування засновані на аналізі поточного стану системи та автоматично адаптують стратегії розподілу навантаження залежно від зміни метрик. Серед метрик, які можуть використовуватися для балансування, кількість активних з'єднань, час останнього відгуку екземпляру, поточна утилізація CPU або RAM. Динамічні методи забезпечують вищий рівень ефективності та відмовостійкості в динамічних умовах, але водночас вимагають наявності моніторингу стану екземплярів. Прикладом є метод найменшого часу відповіді:

$$l(x) = \min(T_x), x = \overline{1, k}, \quad (4.6)$$

де  $l(x)$  – функція для вибору індексу екземпляру;  $T_x$  – час відповіді  $x$ -того екземпляру, а  $k$  – кількість екземплярів

Методи балансування можуть бути націлені на досягнення однієї або кількох цілей при роботі [81]. Зокрема, це можуть бути не тільки метрики QoS, наприклад, час відповіді, а й оптимізація утилізації ресурсів і фінансових витрат. Одноцільові методи націлені на досягненні однієї конкретної цілі, наприклад, мінімізації часу відповіді або оптимізації використання заданого типу ресурсів.

Багатоцільові методи розроблені для оптимізації кількох цілей одночасно, що дозволяє знайти баланс між різними аспектами QoS, такими як час відповіді, пропускна здатність і відмовостійкість. Такі методи дозволяють досягти компроміс між різними цілями для створення збалансованої та ефективної системи.

## 4.2 Розробка інформаційної системи управління обчислювальними ресурсами

### 4.2.1 Розробка функціональної схеми інформаційної системи управління обчислювальними ресурсами

Функціональна структура інформаційної технології, повністю визначається переліком її функцій. На рис. 4.3 зображена функціональна схема системи управління ОР, де кожен окремий елемент функціоналу відображений відповідним відокремленим модулем. Запропонована структура відповідає принципу єдиної відповідальності, дозволяє декомпозувати процеси управління ОР на менші елементи. Кластер є прямим відображенням реальної ІТ-інфраструктури, що включає застосунки, які надають зовнішні та внутрішні послуги. Екземпляри застосунків з множини  $A$  розміщуються на множині вузлів кластера  $N$ , які є відображенням фізичних або віртуальних машин. Користувачі, яким надаються послуги, взаємодіють з кластером, формуючи навантаження на застосунки.

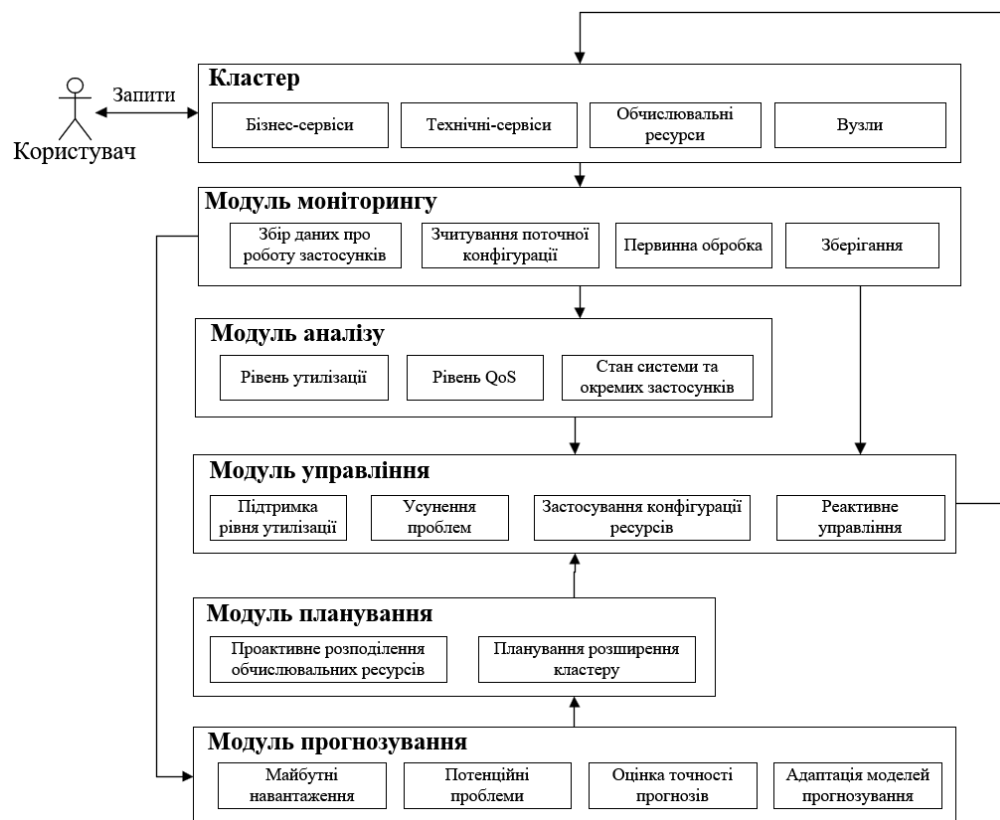


Рисунок 4.3 – Функціональна структура інформаційної технології

#### 4.2.2 Функціональна схема модуля моніторингу

В модулі моніторингу, що зображений на рис. 4.4, відбувається збір, обробка та збереження системних та високорівневих метрик  $P_z$ ,  $z = \overline{1, c}$ , де  $c$  – кількість типів метрик з розміщених застосунків, вузлів та інших компонентів кластера, що включає інформацію про утилізацію ОР та наявний рівень QoS, а також поточної конфігурації розподілу ОР  $X_{ly}$ ,  $l = \overline{1, m}$ ,  $y = \overline{1, h}$ , де  $h$  – кількість типів ресурсів для управління,  $m$  – кількість застосунків.

Для аналізу проблем та тенденцій в середньостроковій та довгостроковій ретроспекціях в модулі моніторингу необхідно реалізувати довгострокове збереження зібраних даних.

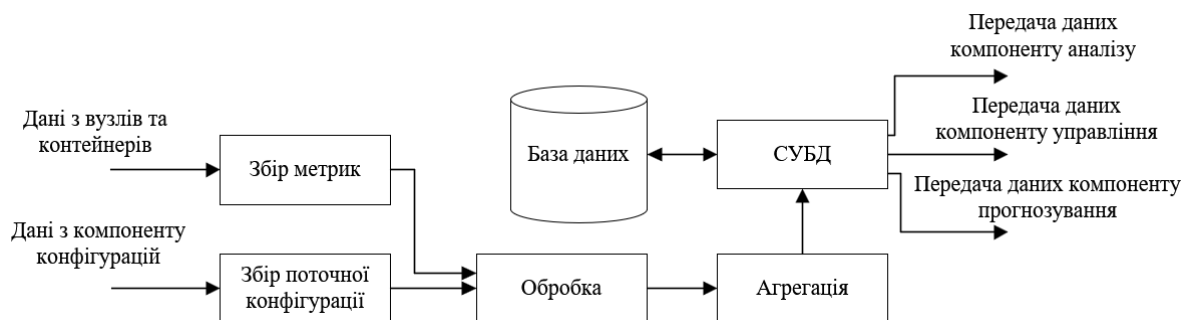


Рисунок 4.4 – Функціональна схема модуля моніторингу

#### 4.2.3 Функціональна схема модуля аналізу

В модулі аналізу, що зображений на рис. 4.5, здійснюється первинний аналіз даних, отриманих в результаті роботи модуля моніторингу.



Рисунок 4.5 – Функціональна схема модуля аналізу

В модулі відбувається аналіз рівня утилізації  $U_{ly}$  на предмет їх достатності та ефективності. Відхилення від запланованих значень свідчить

про необхідність збільшити або зменшити об'єм виділених ресурсів, про що повідомляється модуль управління. Аналогічним чином відбувається аналіз рівня якості послуг та його відповідності SLA або SLO:

$$\Delta Q_l = Q_l - \hat{Q}_l, l = \overline{1, m}, \quad (4.7)$$

де  $\Delta Q_l$  – відхилення рівня якості послуг від цільового відповідно до (1.1), а  $m$  – кількість застосунків. Відхилення від встановлених значень даного показника свідчить про проблеми в системі, які необхідно оперативно усунути.

#### 4.2.4 Функціональна схема модуля прогнозування

В модулі прогнозування, функціональна структура якого зображена на рис. 4.6, здійснює процеси отримання прогнозів на основі історичних даних для планування майбутнього розподілення ресурсів. В модулі прогнозування використовуються методи прогнозування, моделі яких навчаються на доступних даних. Далі ці моделі валідуються на актуальних даних. Моделі з найкращою точністю зберігаються і в подальшому будуть використовуватися для отримання прогнозів використання ОР. Моделі необхідно перенавчати, щоб охопити нові особливості часових рядів робочих навантажень. Результати прогнозування передаються в модуль планування, який на основі отриманих даних оцінює необхідний об'єм кластера в майбутньому і об'єм ресурсів для кожного з розміщених в ньому застосунків.

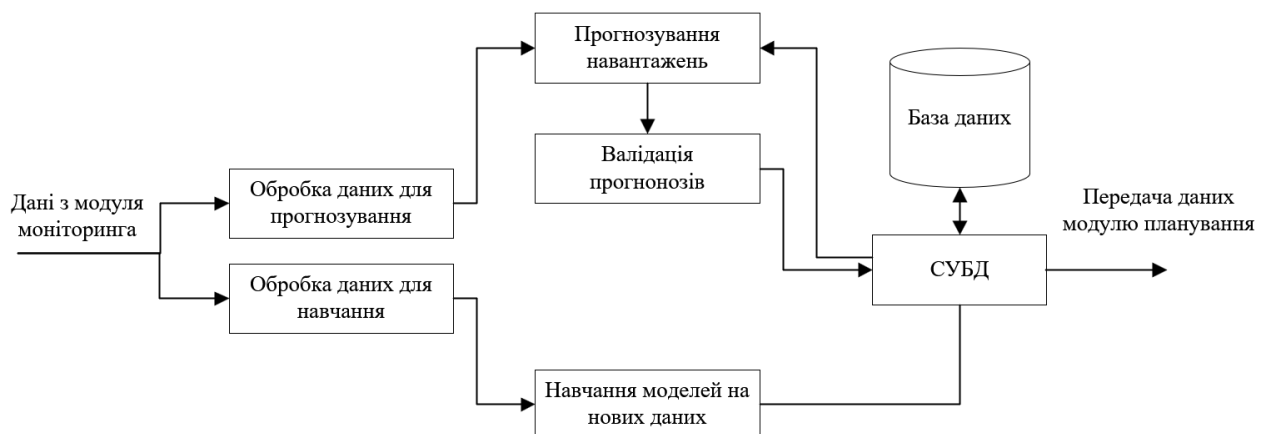


Рисунок 4.6 – Функціональна схема модуля прогнозування



#### 4.2.5 Функціональна схема модуля планування

Модуль планування, зображений на рис. 4.7, відповідає за інтерпретацію отриманих прогнозів. Модуль планування здійснює проактивне управління в інформаційній технології, забезпечуючи передчасну підготовку до зростаючого навантаження як в контексті окремих застосунків, так і в контексті об'єму ресурсів кластера.

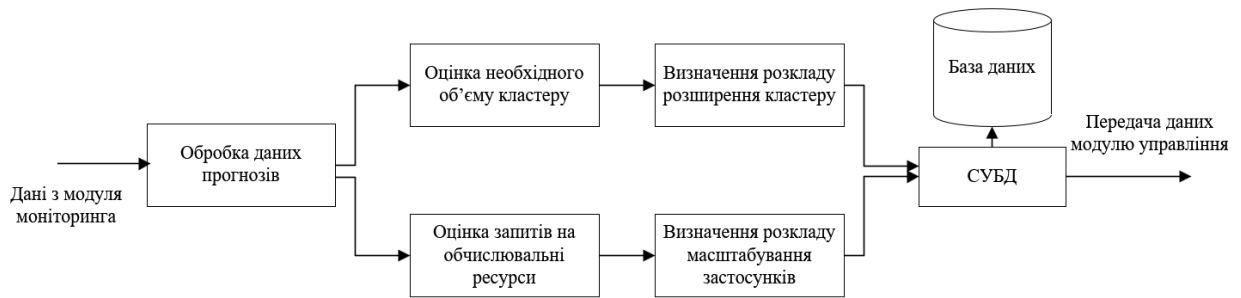


Рисунок 4.7 – Функціональна схема модуля планування

Після формування розкладу виділення ресурсів, команди на виділення додаткових ОР або їх зменшення мають бути надані модулю управління для координації з реактивними компонентами для подальшого застосування. В модулі планування можуть використовуватися методи проактивного управління розроблені в другому та третьому розділі роботи.

#### 4.2.6 Функціональна схема модуля управління

Модуль управління, зображений на рис. 4.8, здійснює як застосування розкладів модуля планування, так і за реактивне управління. Реактивна частина модуля управління здійснює управління конфігурацією виділення ОР для підтримки необхідного рівня QoS та рівня утилізації ресурсів  $U_{ly}$  на основі поточних потреб згідно умов (4.1). Визначення конфігурації ОР для дотримання SLA відбувається саме з метою отримання заданого рівня QoS без врахування фактору ефективності. З іншого боку, функція визначення конфігурації ресурсів для дотримання необхідного рівня утилізації забезпечує ефективне використання виділених ресурсів і обмежує збиткове резервування,

що відповідає вимогам (1.1). Узгодження отриманих конфігурацій відбувається в результаті координації:

$$M_l = f_{coordination}(M_{l,1}, M_{l,2}, \dots, M_{l,q}), l = \overline{1, m}, \quad (4.8)$$

де  $f_{coordination}$  – функція координації,  $M_{l,q}$  – керуючий вплив,  $q$  – кількість керуючих впливів для координування,  $m$  – кількість застосунків.



Рисунок 4.8 – Функціональна схема модуля управління

Також, модуль управління застосовує дані модуля планування, який на основі прогнозів надає оптимальні конфігурації ОР  $X_{ly}$ , які також враховуються при координації. При виникненні помилок в роботі застосунків, причиною яких є некоректна конфігурація ресурсів, даний модуль має адаптувати існуючу конфігурацію для їх усунення.

#### 4.5 Сутність інформаційної технології управління обчислювальними ресурсами

В першу чергу необхідно визначити об'єкт управління, на який здійснюється зовнішній вплив і стан якого відслідковується і корегується шляхом застосування керуючого впливу. Розроблена в дисертації інформаційна технологія призначена для максимізації рівня QoS, що надається застосунком, при мінімізації використання ОР, що виділяються для цього застосунку відповідно до (1.1).

Запити користувачів до  $l$ -го застосунку надходять у кластер з частотою, що описується функцією  $\Omega_l(t)$ . Об'єм ОР, що необхідний для обробки отриманих запитів визначається наступним чином:

$$W_{ly}(t) = f_{ly}(\Omega(l)), l = \overline{1, m}, y = \overline{1, h}, \quad (4.9)$$

де  $W_{ly}(t)$  – об'єм  $y$ -го типу ОР для обробки  $\Omega_l(t)$  запитів,  $f_{ly}(x)$  – функція трансформації частоти запитів в об'єм  $y$ -го ОР для  $l$ -того застосунку,  $m$  – кількість застосунків,  $h$  – кількість екземплярів застосунку. Модуль моніторингу збирає дані про поточне використання ОР екземплярами  $l$ -го застосунку для кожного  $j$ -го екземпляру  $\hat{X}_{lyj}$ , наявний об'єм обчислювальних ресурсів  $X_{lyj}$  та показників рівня якості послуг, які визначаються переліком метрик  $\{P_z\}$ ,  $z = \overline{1, c}$ , де  $c$  – кількість таких метрик. Тоді рівень якості послуг визначається як:

$$Q_l = f_{Q_l}(P_1, P_2, \dots, P_z), l = \overline{1, m}, z = \overline{1, c}, \quad (4.10)$$

де  $f_{Q_l}$  – функція обрахунку рівня QoS для  $l$ -го застосунку, де  $m$  – кількість застосунків,  $c$  – кількість таких метрик. Збір метрик відбувається з визначеною періодичністю  $T_l$ .

В модулі аналізу здійснюється перевірка відповідності наявного рівня якості послуг  $Q_l$  до встановлених адміністратором кластера значень SLA –  $Q'_l$ . Аналогічна перевірка здійснюється для відповідності рівня утилізації ОР, що визначається наступним чином:

$$U_{ly} = \frac{X_{ly}}{\hat{X}_{ly}}, l = \overline{1, m}, y = \overline{1, h}, \quad (4.11)$$

де  $U_{ly}$  – рівень утилізації  $y$ -го типу ОР  $l$ -го застосунку,  $m$  – кількість застосунків,  $h$  – кількість типів ОР. Адміністратор встановлює цільовий показник утилізації ОР –  $U'_{ly}$  для  $y$ -го типу ресурсу  $l$ -го застосунку. У разі

невідповідності поточного рівня якості послуг та утилізації здійснюється комунікація з модулем управління для оперативного усунення проблеми.

В модулі прогнозування здійснюється розрахунок майбутніх значень функції навантаження  $W_{ly}(t)$  на основі історичних даних про фактичне використання ОР застосунками  $\hat{X}_{ly}$ . Використання методів прогнозування, їх навчання та інтерпретація результатів описані в розділі 2.2.

В модулі планування здійснюється розрахунок конфігурації ОР для проактивного управління на основі історичних даних про запити на ОР  $X_{ly}$ , фактичного використання  $\hat{X}_{ly}$  та цільового рівня утилізації  $U'_{ly}$ . Обчислення кількості екземплярів при проактивному управлінні описується (2.14), об'єму ОР кожного екземпляра – (2.15).

В модулі управління здійснюється формування керуючого впливу, що відображено на рис 4.9. В залежності від застосованої адміністратором конфігурації для кожного з застосунків здійснюється реактивне, проактивне або гібридне управління. Реактивне управління ОР описане в (2.14) та (2.15) при  $D_{up}=0$ . Метод інтеграції проактивного і реактивного управління описано в розділі 3.1, а саме (3.1), (3.2) та (3.3). Метод координації вертикального і горизонтального масштабування запропонований в розділі 3.2.

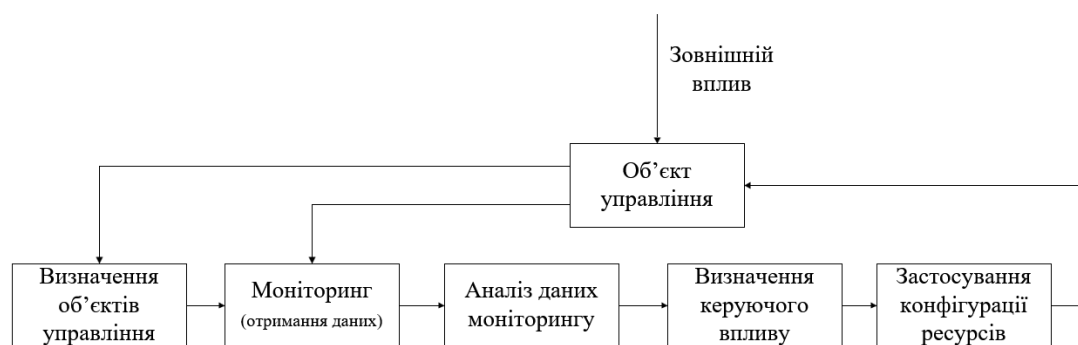


Рисунок 4.9 – Діаграма управління ОР

Застосування керуючого впливу у вигляді кількості екземплярів  $k_l$  та об'єму ОР для кожного з екземплярів  $X_{ly}$ , які є відображенням вертикального і горизонтального компонентів керуючого впливу, відбувається за допомогою комунікації з інтерфейсом кластера.

## 4.4 Особливості використання інформаційної технології управління обчислювальними ресурсами в Kubernetes

### 4.4.1 Особливості управління обчислювальними ресурсами в Kubernetes

Реалізація запропонованої інформаційної технології виконується на основі Kubernetes, з інструментами та обмеженнями даної платформи. Kubernetes надає інструменти для керування розподіленням ОР кластера та можливість базової автоматизації даного процесу. Основною перевагою платформи Kubernetes є її відкритість та універсальність, що дозволяє розширювати функціонал Kubernetes за допомогою власних програмних модулів. Kubernetes надає API для управління кластером, розміщеними застосунками, вузлами та метриками, що значно спрощує реалізацію запропонованої інформаційної технології.

На рис. 4.10 зображена діаграма компонентів кластера Kubernetes з інтегрованою системою управління ОР і відображені зв'язки з вбудованими стандартними компонентами.

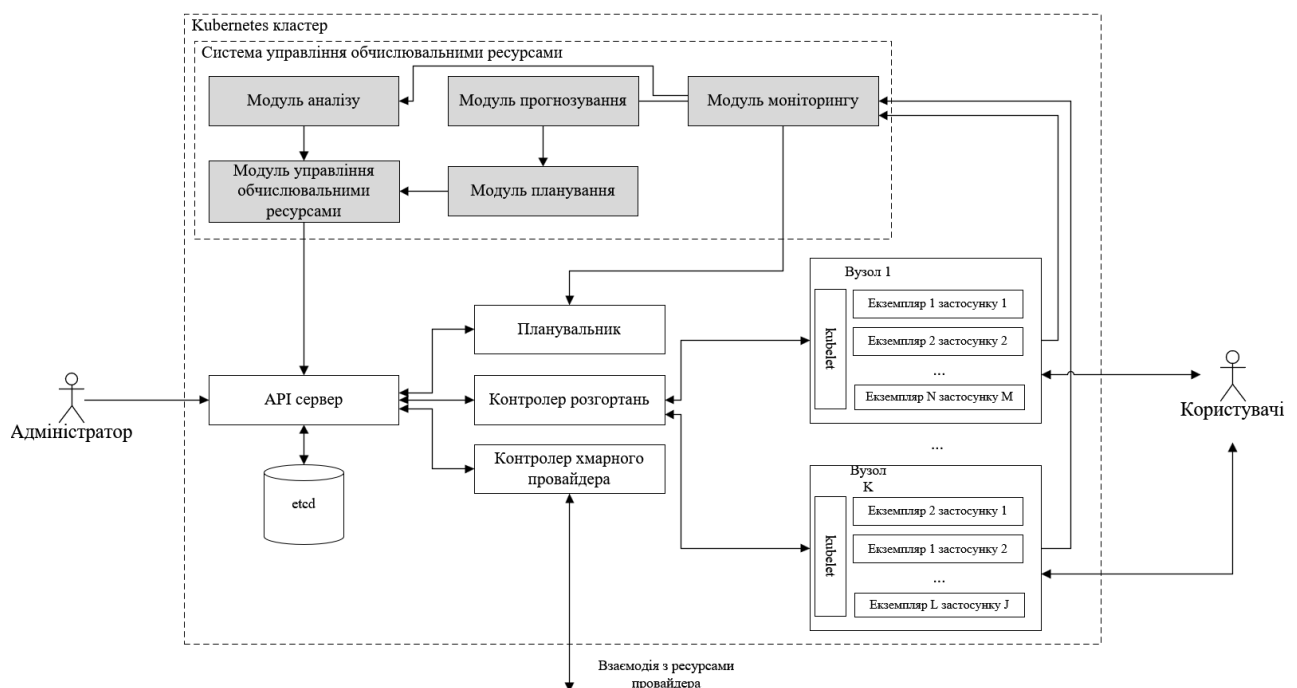


Рисунок 4.10 – Діаграма компонентів кластера з інтегрованою системою управління ОР

Кластер містить вузли, які розміщують корисне навантаження. Дані про використання ресурсів, а також метрики застосунків через агент Kubelet з використанням експортерів надсилаються в модуль моніторингу [82]. Конфігурація ресурсів для застосунків та зміна розміру кластера відбувається через API сервер. Модулі системи управління ОР є технічними застосунками, які також розміщуються на вузлах кластера та можуть бути реалізовані у вигляді контролерів, інтерфейс яких дозволяє гнучко інтегрувати сторонній код в екосистему платформи Kubernetes [82].

#### 4.4.2 Реалізація модулів інформаційної технології

Реалізація модуля моніторингу включає базові компоненти Kubernetes для збору, обробки та збереження метрик застосунків і платформ. Metrics-server є рішенням для збору системних низькорівневих метрик про утилізацію ресурсів вузла розміщеними на ньому контейнеризованими процесами. Metrics-server надає обмежені можливості обробки та запиту метрик, але є надійним і швидким рішенням для вирішення задачі моніторингу. Інсталяція в кластер metrics-server та перевірка його роботи здійснюється за допомогою команди, що зображена на рис. 4.11:

```
git clone https://github.com/kubernetes-sigs/metrics-server.git
cd metrics-server/deploy/kubernetes
kubectl apply -f .
kubectl get deployments metrics-server -n kube-system
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
metrics-server	1/1	1	1	3m

Рисунок 4.11 – Інсталяція metrics-server у кластер

Високорівневі метрики, в тому числі ті, які використовуються для визначення рівня QoS, збираються, обробляються та зберігаються в Prometheus. Використання Prometheus дозволяє збирати метрики з короткотривалих задач за допомогою Pushgateway [83]. Таким чином, Kubernetes надає широкі можливості для збору та зберігання як системних, так і високорівневих метрик.

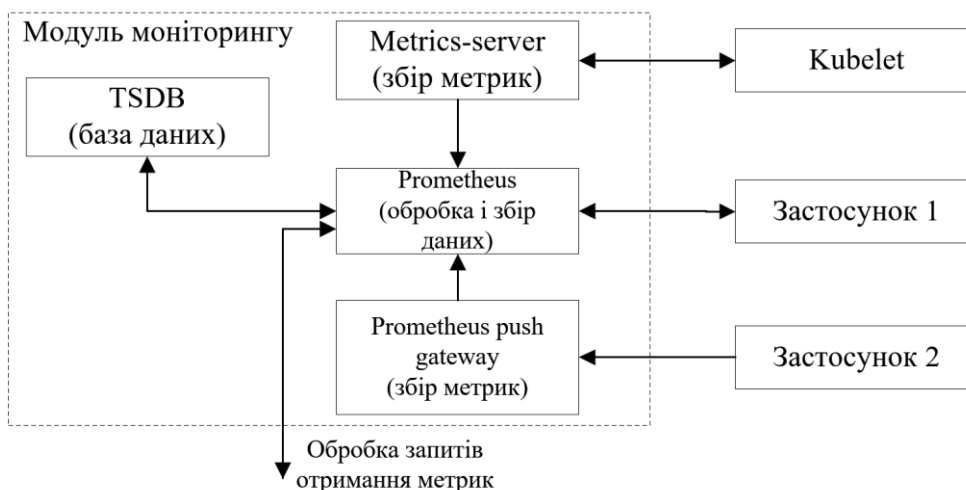


Рисунок 4.12 – Діаграма компонентів модуля моніторингу в Kubernetes

Конфігурація Prometheus, що зображена на рис. 4.13, дозволяє налаштувати період збору метрик з вузлів за допомогою параметру `scrape_interval`. Параметр `evaluation_interval` конфігурує періоду оцінки виконання правил в модулі аналізу. Секція `scrape_configs` необхідна для налаштування правил збору метрик в залежності від типу застосунку. Зокрема підсекція `pushgateway` конфігурує збір метрик за push підходом.

```

global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'pushgateway'
    static_configs:
      - targets: ['pushgateway:9091']
  - job_name: 'kube-state-metrics'
    kubernetes_sd_configs:
      - role: endpoints

```

Рисунок 4.13 – Конфігурація Prometheus

Модуль аналізу також може бути реалізований за допомогою базових компонентів в екосистемі Kubernetes. Alertmanager є рішенням для визначення проблем на основі метрик та сповіщення про них інших компонентів або відповідного персоналу.

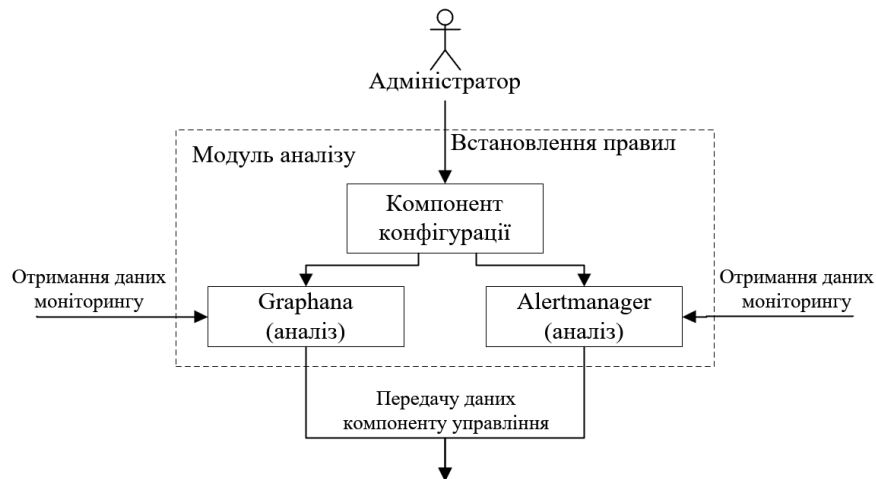


Рисунок 4.14 – Діаграма компонентів модуля аналізу в Kubernetes

Конфігурація Alertmanager рішення складається із запитів до модуля моніторингу, при порушенні яких і відбувається взаємодія з іншими компонентами. Також, Graphana є рішенням не тільки для візуалізації історичних даних, а також для їх аналізу і пошуку потенційних проблем.

На рис. 4.15 зображений приклад візуалізації метрики часу відповіді за допомогою інструменту аналізу Graphana [84]. Зокрема, зображено три показники – 80, 95, 95 перцентилі часу відповіді. За допомогою даного інструменту є можливість аналізувати як системні метрики, так і бізнес-метрики, наприклад, кількість зроблених замовлень.

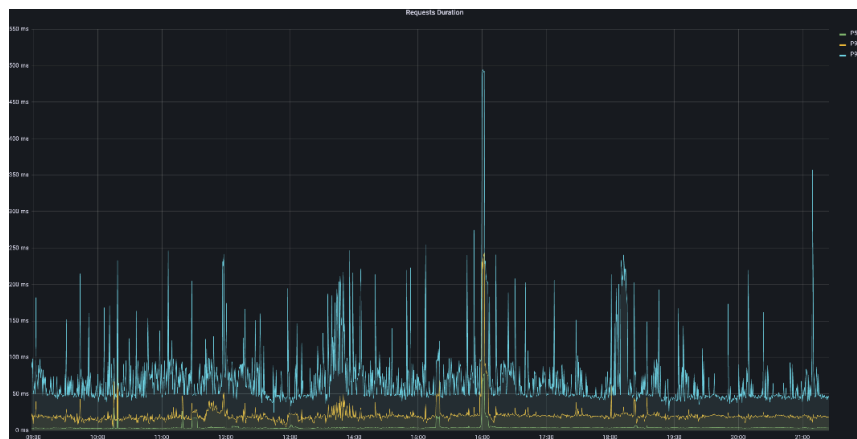


Рисунок 4.15 – Приклад візуалізації метрики часу відповіді в Graphana

На рис. 4.16 зображено приклад конфігурації правил інструменту Alertmanager, які визначають стан застосунку, при якому необхідно надати повідомлення модулю управління. Джерелом даних для Alertmanager є



Prometheus. Параметр конфігурації `expr` включає запит до Prometheus на отримання середньої утилізації процесорного часу. При рівні утилізації більше 80% протягом 5 хвилин, що конфігурується параметром `for`, дані про потенційну проблему передаються компоненту управління. Інструмент Alertmanager здатний надсилати дані про потенційні проблеми як в канали адміністраторів, так і передавати дані по HTTP протоколу вказаним серверним застосунком.

```
groups:
- name: cpu_alerts
  rules:
  - alert: HighCPUUsage
    expr: >
      100 - (avg by (instance) (
        irate(node_cpu_seconds_total{mode="idle"}[5m])
      ) * 100) > 80
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "High CPU usage on {{ $labels.instance }}"
      description: >
        "CPU usage > 80% for more than
        5 minutes on {{ $labels.instance }}"
```

Рисунок 4.16 – Конфігурація правил alertmanager

Конфігурація alertmanager також надає можливість визначати критичність повідомлення за допомогою поля `severity`. Приклад налаштування комунікації alertmanager для передачі повідомлень продемонстрований на рис. 4.17.

```
route:
  receiver: 'default-receiver'
  routes:
  - match:
      severity: 'critical'
      receiver: 'custom-webhook'

receivers:
- name: 'default-receiver'
- name: 'custom-webhook'
  webhook_configs:
  - send_resolved: true
    url: 'https://my-custom-endpoint.example.com/alerts'
```

Рисунок 4.17 – Конфігурації комунікації Alertmanager

Секція `route` вказує зв'язок між правилом alertmanager і отримувачем. В даному прикладі отримувачем є серверний застосунок, для якого вказана адреса.

Модуль прогнозування включає контролер, основною задачею якого є координування взаємодії інших компонентів. Модуль також містить

реалізацію методів прогнозування, які запропоновані в другій частині роботи. Для зберігання даних прогнозування та обраних моделей прогнозування пропонується використовувати S3 або S3-сумісне сховище. Для зберігання також можна використати інструмент, який надає платформа, а саме volumes, проте він є низькорівневим і потребує ручного управління.

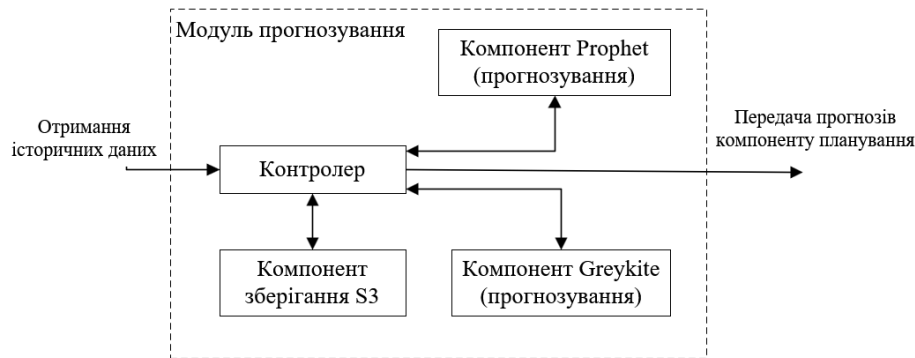


Рисунок 4.18 – Діаграма компонентів модуля прогнозування в Kubernetes

Дані для прогнозування отримуються шляхом запиту до Prometheus. На рис. 4.19 продемонстровано приклад розкладання часового ряду за допомогою бібліотеки Prophet на компоненти сезонності і часового ряду. При прогнозуванні бібліотека використовує описані компоненти для отримання точних результатів.

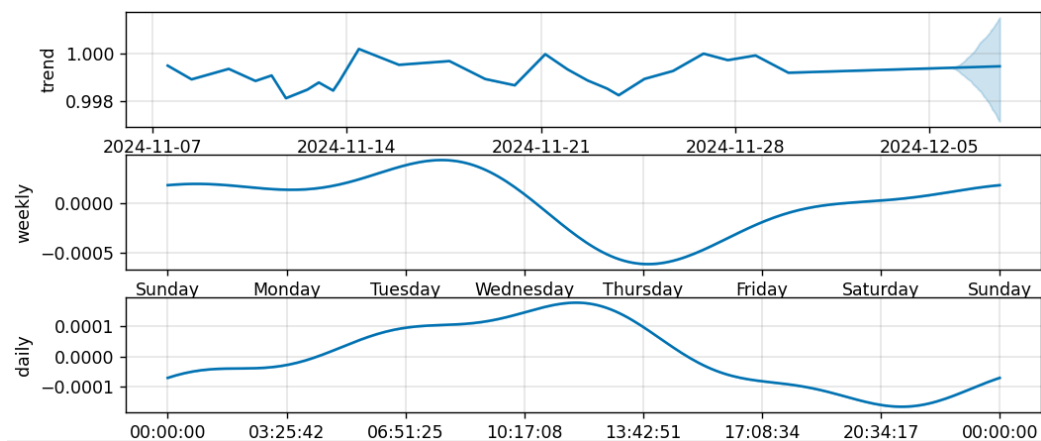


Рисунок 4.19 – Аналіз часового рядку за допомогою Prophet

Модуль планування будується на основі інструменту CronJobs, який дозволяє відкласти виконання завдання до вказаного моменту часу. Механізм відкладених завдань є корисним, якщо відомо про зростання навантаження в

певний період часу та дозволяє завчасно відправити команду модулю управління про необхідність збільшення кількості ОР для застосунків. Також, модуль планування включає два програмні компоненти. Перший компонент відповідає за додавання або видалення вузлів кластера відповідно до наданого прогнозу, а другий – за масштабування конфігурації застосунків.

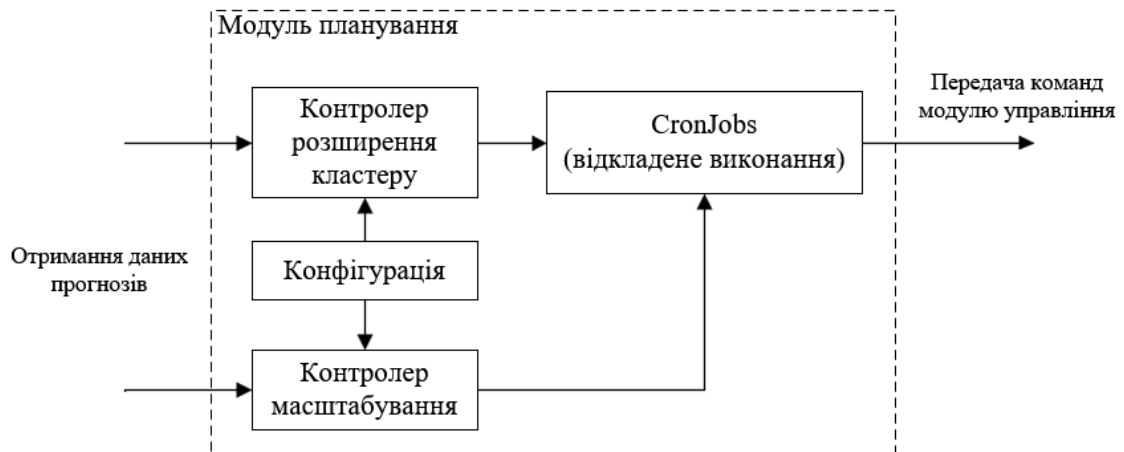


Рисунок 4.20 – Діаграма компонентів модуля планування в Kubernetes

На рис. 4.21 продемонстровано конфігурацію відкладених задач CronJobs, що є частиною екосистеми Kubernetes. Поле `schedule` визначає розклад, коли необхідно виконати дану задачу [82].

```

apiVersion: batch/v1
kind: CronJob
metadata:
  name: example-cronjob
  namespace: default
spec:
  schedule: "* / 5 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: example-task
              image: busybox:latest
              command: ["sh", "-c", "echo 'Running scheduled task at $(date)'; sleep 10"]

```

Рисунок 4.21 – Конфігурація CronJobs

Поле `command` визначає команду ОС для виконання, в якій має бути звернення до модуля управління з передачею оптимальної конфігурації ОР для подальшого масштабування.

Компонент модуля управління, що зображений на рис. 4.22 складається з сторонніх програмних модулів. Контролер реактивного управління відповідає за розрахунок поточної потреби в ОР на основі поточного стану, що можна реалізувати на основі наявних рішень для реактивного масштабування НРА та VPA. Контролер проактивного управління отримує команди модуля прогнозування, проводить оцінку доцільності застосування. Компонент координації необхідний для узгодження реактивного і проактивного компонентів, оскільки неузгоджені рішення можуть призвести до порушень в роботі системи. Метод координування вертикального і горизонтального масштабування описаний (3.4). Метод координування реактивного і проактивного описаний в розділі 3.1.

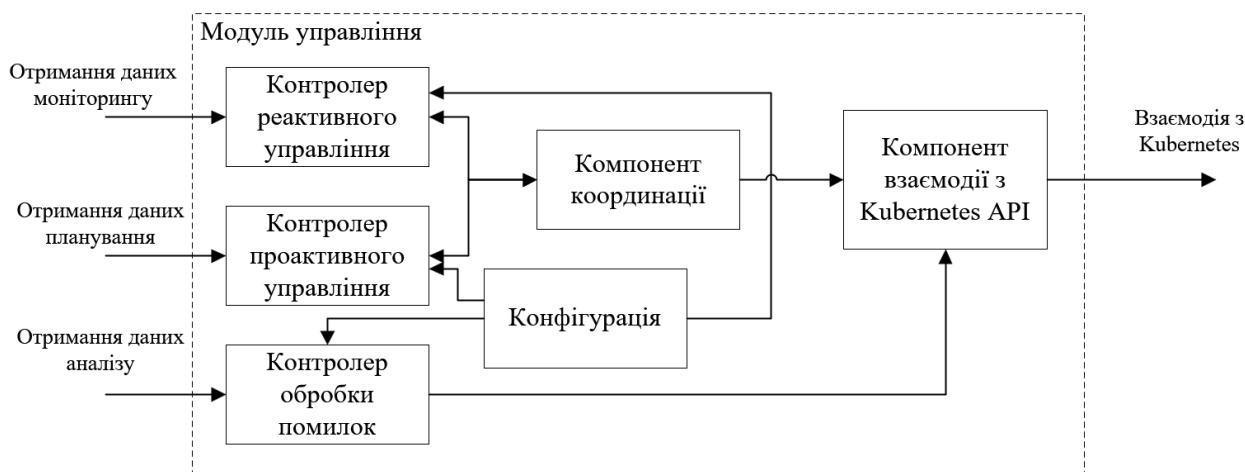


Рисунок 4.22 – Діаграма компонентів модуля управління в Kubernetes

Компонент конфігурації реалізовується використанням внутрішнього ресурсу Kubernetes – ConfigMap. Контролер обробки помилок необхідний для швидкого реагування на виникаючі проблеми, зокрема, помилки нестачі пам'яті. Для взаємодії з ресурсами Kubernetes пропонується застосовувати існуючі клієнти, зокрема, kubectl.

Для реактивного масштабування Kubernetes надає відповідний інструмент НРА, конфігурація якого зображена на рис. 4.23. В полі `type.name` вказується тип ОР, який є пріоритетним при розрахунку кількості екземплярів. Поле `averageUtilization` є цільовим значенням утилізації вказаного типу ОР.

Можуть вказуватися поля `minReplicas` і `maxReplicas` для обмеження мінімальної і максимальної кількості екземплярів відповідно.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
...
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    name: example-deployment
  metrics:
    - type: Resource
      ...
        name: cpu
        target:
          averageUtilization: 80
```

Рисунок 4.23 – Конфігурація НРА

Конфігурація контролера проактивного масштабування, що запропонований в розділі 2.3, додатково включає поля `upscalingTime`, `accuracyThreshold`, `seasonalitySeconds`, що зображено на рис. 4.24. Поле `upscalingTime` описує час розгортання нового екземпляра застосунку для точного обчислення часу масштабування при збільшенні навантаження. Поле `accuracyThreshold` є критичним значенням точності прогнозів, при якому подальше застосування проактивного контролера не є ефективним.

```
metadata:
  name: example-proactive-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: example-deployment
  targetCPUUtilization: 75
  upscalingTime: "5m"
  accuracyThreshold: 0.1
  seasonalitySeconds: [3600, 86400, 604800]
```

Рисунок 4.24 – Конфігурація проактивного масштабувача

На рис. 4.25 зображено приклад роботи компоненти горизонтального масштабування, отриманий за допомогою описаних модулів моніторингу і аналізу.

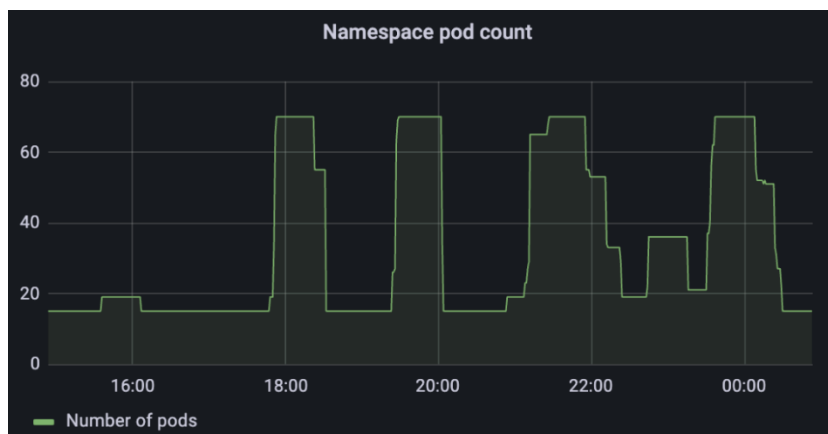


Рисунок 4.25 – Приклад горизонтального масштабування застосунку

Для імплементації власних програмних модулів в Kubernetes необхідно взаємодіяти з API. Для цього можна використати спеціалізовані бібліотеки, які надають додатковий рівень абстракції між конфігурацією кластера і програмним кодом. На рис. 4.26 зображений приклад роботи з Kubernetes API з використанням мови програмування Python для оновлення запитів на ОР для вказаного розгортання.

```
requests = {"cpu": "100m", "memory": "128Mi"}
limits = {"cpu": "500m", "memory": "256Mi"}

container = client.V1Container(
    name="example-container",
    image="nginx:latest",
    resources=client.V1ResourceRequirements(requests=requests, limits=limits),
)

api = client.AppsV1Api()
response = api.create_namespaced_deployment(
    body=deployment,
    namespace="default"
)
```

Рисунок 4.26 – Приклад взаємодії з Kubernetes API з використанням мови програмування Python

Приклад коду взаємодії включає створення конфігурації контейнера, для якого задаються запити та ліміти на ОР за допомогою параметрів `requests` та `limits` відповідно. Конфігурація контейнера передається на API Kubernetes та застосовується для всіх екземплярів розгортання.

## Висновки до розділу 4

Розроблено інформаційну технологію управління ОР на основі запропонованих в дисертації моделей та методів. Інформаційна технологія спрямована на забезпечення встановленого рівня QoS з ефективним використання ОР. Запропоновано політики для вирішення ситуацій при перевищенні запитів на ОР. Описано процеси резервування загального об'єму ОР кластеру та його масштабування. Визначено структуру інформаційної технології, на основі якої створено ІС управління ОР з декомпозицією функціоналу в окремих модулях.

На основі запропонованої інформаційної технології управління ОР розроблено СУ ОР для Kubernetes, що враховує наявні інструменти для управління ОР, зокрема, Prometheus, metrics-server, HPA, VPA та спеціальні контролери та ресурси. Продемонстровано конфігурацію компонентів ІС для роботи в Kubernetes.

Таким чином, запропонована інформаційна технологія надає комплексний підхід до управління ОР, спрямований на забезпечення заданого рівня QoS при дотриманні визначеного рівня утилізації ОР, надійності та масштабованості ІС в динамічних умовах роботи.

## ВИСНОВКИ

У дисертаційній роботі вирішується актуальна проблема управління IT-інфраструктурою – автоматизація управління ОР з врахуванням потреб застосунків та встановлених вимог до якості послуг. Поставлені в дисертаційній роботі задачі виконані в повному обсязі.

На основі проведеного аналізу поточного стану і тенденцій керування IT-інфраструктурою встановлено, що використання контейнеризації та віртуалізації, хмарних платформ і мікросервісної архітектури для проектування ІС вимагають створення нових методів для керування ОР з врахуванням наявності великою кількості застосунків в межах однієї ІС для забезпечення необхідного рівня якості послуг.

Доведено, що реактивні методи управління ОР, які широко застосовуються при проектуванні ІС, не можуть надавати консистентний рівень якості послуг через затримки отримання даних та розгортання нових екземплярів. Обґрунтовано необхідність розробки та впровадження проактивних та гібридних методів масштабування, які дозволяють мінімізувати збиткове резервування ОР при забезпеченні встановленого рівня якості послуг.

Розроблено метод комбінованого прогнозування, який відрізняється наявністю компонентів короткострокового і довгострокового прогнозування, що дозволяє оцінити масштаб аномальних навантажень за допомогою зіставлення з шаблонами попередніх наявних в історичних даних. Експериментальне дослідження демонструє підвищення точності прогнозів до 95% у порівнянні з базовими методами.

Запропоновано метод проактивного масштабування, який відрізняється підтримкою робочих навантажень, що включають сезонності та тенденції. Метод дозволяє масштабувати застосунки до зростання навантаження резервуючи мінімально необхідний об'єм ОР задля дотримання вимог SLA. Результати експериментальних досліджень доводять зменшення резервування



ОР на 47% у порівнянні зі статичним підходом виділення ОР при аналогічному часу відповіді та зменшення часу відповіді з 166 мс до 23 мс у порівнянні з реактивним масштабуванням.

Запропоновано гібридний проактивно-реактивний метод масштабування для забезпечення роботи застосунків в умовах низької точності прогнозів, відсутності даних або їх низької якості для забезпечення роботи застосунків в умовах аномальних навантажень.

Запропоновано метод інтеграції вертикального і горизонтального масштабування, який відрізняється наявністю координатора для узгодження вертикального і горизонтального управління, що дозволяє зменшити збиткове резервування ОР. Застосування даного методу дозволяє зменшити резервування процесорного часу на 65% у порівнянні з підходом без масштабування.

Вдосконалена ІТ управління ОР, яка відрізняється наявністю модулів проактивного та гібридного управління, комплексно описує підхід до управління ОР в кластері Kubernetes, включаючи моніторинг стану системи, аналіз даних та пошук потенційних проблем, прогнозування навантаження, оцінку доцільності застосування оновленої конфігурації та взаємодії компонентів ІТ з кластером. Розроблені методи є ключовою частиною даної ІТ. Практична цінність даної ІТ полягає в її повній реалізації в кластері Kubernetes за допомогою вбудованих інструментів даної платформи.

Для запропонованих в роботі методів реалізовані програмні модулі та проведені експериментальні дослідження їх ефективності. Результати досліджень довели ефективність запропонованих рішень.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ролік, О. І., Теленик, С. Ф., Ясочка, М. В. (2018). Управління корпоративною ІТ-інфраструктурою. Наукова думка.
2. Ardagna, D., Casale, G., Ciavotta, M., Pérez, J. F., & Wang, W. (2014). Quality-of-service in cloud computing: modeling techniques and their applications. In *Journal of Internet Services and Applications* (Vol. 5, Issue 1). Sociedade Brasileira de Computacao - SB. <https://doi.org/10.1186/s13174-014-0011-3>
3. Sun, Y., White, J., Eade, S., & Schmidt, D. C. (2016). ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization. In *Journal of Systems and Software* (Vol. 116, pp. 146–161). Elsevier BV. <https://doi.org/10.1016/j.jss.2015.08.006>
4. Truyen, E., Van Landuyt, D., Preuveneers, D., Lagaisse, B., & Joosen, W. (2019). A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks. In *Applied Sciences* (Vol. 9, Issue 5, p. 931). MDPI AG. <https://doi.org/10.3390/app9050931>
5. Rejiba, Z., & Chamanara, J. (2022). Custom Scheduling in Kubernetes: A Survey on Common Problems and Solution Approaches. In *ACM Computing Surveys* (Vol. 55, Issue 7, pp. 1–37). Association for Computing Machinery (ACM). <https://doi.org/10.1145/3544788>
6. Khan, A. (2017). Key Characteristics of a Container Orchestration Platform to Enable a Modern Application. In *IEEE Cloud Computing* (Vol. 4, Issue 5, pp. 42–48). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/mcc.2017.4250933>
7. Gill, S. S., & Buyya, R. (2020). Failure Management for Reliable Cloud Computing: A Taxonomy, Model, and Future Directions. In *Computing in Science & Engineering* (Vol. 22, Issue 3, pp. 52–63). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/mcse.2018.2873866>
8. Tao, Z., Xia, Q., Hao, Z., Li, C., Ma, L., Yi, S., & Li, Q. (2019). A Survey of Virtual Machine Management in Edge Computing. In *Proceedings of the IEEE* (Vol.

- 107, Issue 8, pp. 1482–1499). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/jproc.2019.2927919>
9. Hilley, D. (2009). Cloud Computing: A Taxonomy of Platform and Infrastructure-level Offerings.
10. Carrión, C. (2022). Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. In *ACM Computing Surveys* (Vol. 55, Issue 7, pp. 1–37). Association for Computing Machinery (ACM). <https://doi.org/10.1145/3539606>
11. Thurgood, B., & Lennon, R. G. (2019). Cloud Computing With Kubernetes Cluster Elastic Scaling. In *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems* (pp. 1–7). ICFNDS '19: 3rd International Conference on Future Networks and Distributed Systems. ACM. <https://doi.org/10.1145/3341325.3341995>
12. Koukis, G., Skaperas, S., Kapetanidou, I. A., Mamatas, L., & Tsaoussidis, V. (2024). Performance Evaluation of Kubernetes Networking Approaches across Constraint Edge Environments. *arXiv*. <https://doi.org/10.48550/ARXIV.2401.07674>
13. Ролік, А.І. (2012). Концепція управління корпоративною ІТ-інфраструктурою. *Вісник Національного технічного університету України "КПІ". Інформатика, управління та обчислювальна техніка* (Вип. 56, с. 31-55).
14. Ролік, А.І. (2013). Управління рівнем послуг корпоративної ІТ-інфраструктури на основі координатора. *Вісник Національного технічного університету України "КПІ". Інформатика, управління та обчислювальна техніка* (Вип. 59, с. 98–105).
15. Carrión, C. (2022). Kubernetes SchAbdollahi Vayghan, L., Saied, M. A., Toeroe, M., & Khendek, F. (2018). Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (pp. 970–973). IEEE. <https://doi.org/10.1109/cloud.2018.00148>
16. Watada, J., Roy, A., Kadikar, R., Pham, H., & Xu, B. (2019). Emerging Trends, Techniques and Open Issues of Containerization: A Review. In *IEEE Access*

- (Vol. 7, pp. 443–472). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/access.2019.2945930>
17. Vayghan, L. A., Saied, M. A., Toeroe, M., & Khendek, F. (2021). A Kubernetes controller for managing the availability of elastic microservice based stateful applications. In *Journal of Systems and Software* (Vol. 175, p. 110924). Elsevier BV. <https://doi.org/10.1016/j.jss.2021.110924>
  18. Ward, J. S., & Barker, A. (2014). Observing the clouds: a survey and taxonomy of cloud monitoring. In *Journal of Cloud Computing* (Vol. 3, Issue 1). Springer Science and Business Media LLC. <https://doi.org/10.1186/s13677-014-0024-2>
  19. Hasselmeyer, P., & d’Heureuse, N. (2010). Towards holistic multi-tenant monitoring for virtual data centers. In *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*. IEEE. <https://doi.org/10.1109/nomsw.2010.5486528>
  20. Katsaros, G., Kübert, R., & Gallizo, G. (2011). Building a Service-Oriented Monitoring Framework with REST and Nagios. In *2011 IEEE International Conference on Services Computing*. 2011 IEEE International Conference on Services Computing (SCC). IEEE. <https://doi.org/10.1109/scc.2011.53>
  21. Walters, J. P., Chaudhary, V., Cha, M., Guercio, S., & Gallo, S. (2008). A Comparison of Virtualization Technologies for HPC. In *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*. IEEE. <https://doi.org/10.1109/aina.2008.45>
  22. Thiyyakat, M., Kalambur, S., & Sitaram, D. (2020). Improving Resource Isolation of Critical Tasks in a Workload. In *Job Scheduling Strategies for Parallel Processing* (pp. 45–67). Springer International Publishing. [https://doi.org/10.1007/978-3-030-63171-0\\_3](https://doi.org/10.1007/978-3-030-63171-0_3)
  23. Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015). An updated performance comparison of virtual machines and Linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. <https://doi.org/10.1109/ispass.2015.7095802>

24. Kim, J., Shin, P., Noh, S., Ham, D., & Hong, S. (2018). Reducing Memory Interference Latency of Safety-Critical Applications via Memory Request Throttling and Linux Cgroup. In 2018 31st IEEE International System-on-Chip Conference (SOCC) (pp. 215–220). IEEE. <https://doi.org/10.1109/socc.2018.8618555>
25. Muzumdar, P., Bhosale, A., Basyal, G. P., & Kurian, G. (2024). Navigating the Docker Ecosystem: A Comprehensive Taxonomy and Survey. arXiv. <https://doi.org/10.48550/ARXIV.2403.17940>
26. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., & Warfield, A. (2003). Xen and the art of virtualization. In Proceedings of the nineteenth ACM symposium on Operating systems principles. ACM. <https://doi.org/10.1145/945445.945462>
27. Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2019). Cloud Container Technologies: A State-of-the-Art Review. In IEEE Transactions on Cloud Computing (Vol. 7, Issue 3, pp. 677–692). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/tcc.2017.2702586>
28. Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., & Zhou, W. (2018). A Comparative Study of Containers and Virtual Machines in Big Data Environment. arXiv. <https://doi.org/10.48550/ARXIV.1807.01842>
29. Mustafa, S., Nazir, B., Hayat, A., Khan, A. ur R., & Madani, S. A. (2015). Resource management in cloud computing: Taxonomy, prospects, and challenges. In Computers & Electrical Engineering (Vol. 47, pp. 186–203). Elsevier BV. <https://doi.org/10.1016/j.compeleceng.2015.07.021>
30. Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2018). Elasticity in Cloud Computing: State of the Art and Research Challenges. In IEEE Transactions on Services Computing (Vol. 11, Issue 2, pp. 430–447). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/tsc.2017.2711009>
31. Straesser, M., Grohmann, J., von Kistowski, J., Eismann, S., Bauer, A., & Kounev, S. (2022). Why Is It Not Solved Yet? In Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering (pp. 105–115). ACM. <https://doi.org/10.1145/3489525.3511680>

32. Omelchenko, V.V., & Rolik, O.I., (2022). Automation of resource management in information systems based on reactive vertical scaling. In *Adaptive Systems of Automatic Control* (Vol. 2, Issue 41, pp. 65–78). Kyiv Politechnic Institute. <https://doi.org/10.20535/1560-8956.41.2022.271344>
33. Rolik, O. I., & Omelchenko, V. V. (2024). Proactive horizontal scaling method for Kubernetes. In *Radio Electronics, Computer Science, Control* (Issue 1, p. 221). National University Zaporizhzhia Polytechnic; <https://doi.org/10.15588/1607-3274-2024-1-20>
34. Maurer, M., Breskovic, I., Emeakaroha, V. C., & Brandic, I. (2011). Revealing the MAPE loop for the autonomic management of Cloud infrastructures. In *2011 IEEE Symposium on Computers and Communications (ISCC)* (pp. 147–152). IEEE. <https://doi.org/10.1109/iscc.2011.5984008>
35. Lehrig, S., Eikerling, H., & Becker, S. (2015). Scalability, Elasticity, and Efficiency in Cloud Computing. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures* (pp. 83–92). *CompArch '15: Federated Events on Component-Based Software Engineering and Software Architecture*. ACM. <https://doi.org/10.1145/2737182.2737185>
36. Jun Wu, & Tei-Wei Kuo. (n.d.). Real-time scheduling of CPU-bound and I/O-bound processes. In *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99* (pp. 303–310. IEEE Comput. Soc. <https://doi.org/10.1109/rtsa.1999.811262>
37. Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., & Kim, S. (2020). Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration. In *Sensors* (Vol. 20, Issue 16, p. 4621). MDPI AG. <https://doi.org/10.3390/s20164621>
38. Omelchenko, V. V., Rolik, O. I. (2023). Workloads prediction methods for proactive resource scaling in Kubernetes. *III International Scientific Symposium “Intelligent Solutions” (IntSol-2023)*.
39. Omelchenko, V. V., Rolik, O. I. (2023). Forecasting-at-scale algorithms for prediction cluster workload. *The International Conference on Security, Fault Tolerance, Intelligence*.

40. Qu, C., Calheiros, R. N., & Buyya, R. (2018). Auto-Scaling Web Applications in Clouds. In *ACM Computing Surveys* (Vol. 51, Issue 4, pp. 1–33). Association for Computing Machinery (ACM). <https://doi.org/10.1145/3148149>
41. Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. In *Journal of Grid Computing* (Vol. 12, Issue 4, pp. 559–592). Springer Science and Business Media LLC. <https://doi.org/10.1007/s10723-014-9314-7>
42. Caron, E., Desprez, F., & Muresan, A. (2011). Pattern Matching Based Forecast of Non-periodic Repetitive Behavior for Cloud Clients. In *Journal of Grid Computing* (Vol. 9, Issue 1, pp. 49–64). Springer Science and Business Media LLC. <https://doi.org/10.1007/s10723-010-9178-4>
43. Omelchenko, V. V., Rolik, O. I. (2024). Combined forecasting method for cloud workloads. *Problems of Infocommunications. Science and Technology*” (PIC S&T'2024)
44. Calheiros, R. N., Masoumi, E., Ranjan, R., & Buyya, R. (2015). Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. In *IEEE Transactions on Cloud Computing* (Vol. 3, Issue 4, pp. 449–458). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/tcc.2014.2350475>
45. Zhenhuan Gong, Xiaohui Gu, & Wilkes, J. (2010). PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *2010 International Conference on Network and Service Management*. 2010 International Conference on Network and Service Management (CNSM). IEEE. <https://doi.org/10.1109/cnsm.2010.5691343>
46. Zhong, J., Duan, S., & Li, Q. (2019). Auto-Scaling Cloud Resources using LSTM and Reinforcement Learning to Guarantee Service-Level Agreements and Reduce Resource Costs. In *Journal of Physics: Conference Series* (Vol. 1237, Issue 2, p. 022033). IOP Publishing. <https://doi.org/10.1088/1742-6596/1237/2/022033>
47. De Livera, A. M., Hyndman, R. J., & Snyder, R. D. (2011). Forecasting Time Series With Complex Seasonal Patterns Using Exponential Smoothing. In *Journal*

- of the American Statistical Association (Vol. 106, Issue 496, pp. 1513–1527). Informa UK Limited. <https://doi.org/10.1198/jasa.2011.tm09771>
48. Amekraz, Z., & Hadi, M. Y. (2018). Higher order statistics based method for workload prediction in the cloud using ARMA model. In 2018 International Conference on Intelligent Systems and Computer Vision (ISCV) (pp. 1–5). 2018. IEEE. <https://doi.org/10.1109/isacv.2018.8354078>
  49. Taylor, S. J., & Letham, B. (2017). Forecasting at scale. PeerJ. <https://doi.org/10.7287/peerj.preprints.3190v2>
  50. Umlauf, N., Adler, D., Kneib, T., Lang, S., & Zeileis, A. (2015). Structured Additive Regression Models: An R Interface to BayesX. In Journal of Statistical Software (Vol. 63, Issue 21). Foundation for Open Access Statistics. <https://doi.org/10.18637/jss.v063.i2>
  51. Triebe, O., Hewamalage, H., Pilyugina, P., Laptev, N., Bergmeir, C., & Rajagopal, R. (2021). NeuralProphet: Explainable Forecasting at Scale. arXiv. <https://doi.org/10.48550/ARXIV.2111.15397>
  52. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. arXiv. <https://doi.org/10.48550/ARXIV.1406.2661>
  53. Shu, Y. (2005). Wireless Traffic Modeling and Prediction Using Seasonal ARIMA Models. In IEICE Transactions on Communications (Vols. E88-B, Issue 10, pp. 3992–3999). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1093/ietcom/e88-b.10.3992>
  54. Wang, T., Wei, J., Zhang, W., Zhong, H., & Huang, T. (2014). Workload-aware anomaly detection for Web applications. In Journal of Systems and Software (Vol. 89, pp. 19–32). Elsevier BV. <https://doi.org/10.1016/j.jss.2013.03.060>
  55. Hagemann, T., & Katsarou, K. (2020). A Systematic Review on Anomaly Detection for Cloud Computing Environments. In 2020 3rd Artificial Intelligence and Cloud Computing Conference (pp. 83–96). AICCC 2020: 2020 3rd Artificial Intelligence and Cloud Computing Conference. ACM. <https://doi.org/10.1145/3442536.3442550>



56. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection. In *ACM Computing Surveys* (Vol. 41, Issue 3, pp. 1–58). Association for Computing Machinery (ACM). <https://doi.org/10.1145/1541880.1541882>
57. Malhotra, P., Vig, L., Shroff, G., Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*
58. Pena, E. H. M., de Assis, M. V. O., & Proenca, M. L. (2013). Anomaly Detection Using Forecasting Methods ARIMA and HWDS. In *2013 32nd International Conference of the Chilean Computer Science Society (SCCC)* (pp. 63–66). IEEE. <https://doi.org/10.1109/sccc.2013.18>
59. You, J., Cela, A., Natowicz, R., Ouanounou, J., & Siarry, P. (2024). Anomaly Prediction: A Novel Approach with Explicit Delay and Horizon. *arXiv*. <https://doi.org/10.48550/ARXIV.2408.04377>
60. Van Onsem, M., De Paepe, D., Vanden Haute, S., Bonte, P., Ledoux, V., Lejon, A., Ongenae, F., Dreesen, D., & Van Hoecke, S. (2022). Hierarchical pattern matching for anomaly detection in time series. In *Computer Communications* (Vol. 193, pp. 75–81). Elsevier BV. <https://doi.org/10.1016/j.comcom.2022.06.027>
61. Caron, E., Desprez, F., & Muresan, A. (2011). Pattern Matching Based Forecast of Non-periodic Repetitive Behavior for Cloud Clients. In *Journal of Grid Computing* (Vol. 9, Issue 1, pp. 49–64). Springer Science and Business Media LLC. <https://doi.org/10.1007/s10723-010-9178-4>
62. Li, H., Liu, J., Yang, Z., Liu, R. W., Wu, K., & Wan, Y. (2020). Adaptively constrained dynamic time warping for time series classification and clustering. In *Information Sciences* (Vol. 534, pp. 97–116). Elsevier BV. <https://doi.org/10.1016/j.ins.2020.04.009>
63. Gill, P. E., & Wong, E. (2011). Sequential Quadratic Programming Methods. In *The IMA Volumes in Mathematics and its Applications* (pp. 147–224). Springer New York. [https://doi.org/10.1007/978-1-4614-1927-3\\_6](https://doi.org/10.1007/978-1-4614-1927-3_6)

64. Shen, Z., Subbiah, S., Gu, X., & Wilkes, J. (2011). CloudScale. In Proceedings of the 2nd ACM Symposium on Cloud Computing. SOCC '11: ACM Symposium on Cloud Computing in conjunction with SOSP 2011. ACM. <https://doi.org/10.1145/2038916.2038921>
65. Iqbal, W., Dailey, M. N., Carrera, D., & Janecek, P. (2011). Adaptive resource provisioning for read intensive multi-tier applications in the cloud. In Future Generation Computer Systems (Vol. 27, Issue 6, pp. 871–879). Elsevier BV. <https://doi.org/10.1016/j.future.2010.10.016>
66. Ali-Eldin, A., Kihl, M., Tordsson, J., & Elmroth, E. (2012). Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In Proceedings of the 3rd workshop on Scientific Cloud Computing (pp. 31–40). HPDC'12: The 21st International Symposium on High-Performance Parallel and Distributed Computing. ACM. <https://doi.org/10.1145/2287036.2287044>
67. Ali-Eldin, A., Tordsson, J., & Elmroth, E. (2012). An adaptive hybrid elasticity controller for cloud infrastructures. In 2012 IEEE Network Operations and Management Symposium (pp. 204–212). 2012 IEEE/IFIP Network Operations and Management Symposium (NOMS 2012). IEEE. <https://doi.org/10.1109/noms.2012.6211900>
68. Bergmeir, C., Costantini, M., & Benítez, J. M. (2014). On the usefulness of cross-validation for directional forecast evaluation. In Computational Statistics Data Analysis (Vol. 76, pp. 132–143). Elsevier BV. <https://doi.org/10.1016/j.csda.2014.02.001>
69. Omelchenko, V. V., Rolik, O. I. (2025). Hybrid method for horizontal and vertical computational resource scaling. In Advanced Information Technology (Vol. 1, pp. 1–10).
70. Rochman, Y., Levy, H., & Brosh, E. (2014). Efficient resource placement in cloud computing and network applications. In ACM SIGMETRICS Performance Evaluation Review (Vol. 42, Issue 2, pp. 49–51). Association for Computing Machinery (ACM). <https://doi.org/10.1145/2667522.2667538>


71. Dutta, S., Gera, S., Verma, A., & Viswanathan, B. (2012). SmartScale: Automatic Application Scaling in Enterprise Clouds. In 2012 IEEE Fifth International Conference on Cloud Computing. 2012 IEEE 5th International Conference on Cloud Computing (CLOUD). IEEE. <https://doi.org/10.1109/cloud.2012.12>
72. Incerto, E., Tribastone, M., & Trubiani, C. (2018). Combined Vertical and Horizontal Autoscaling Through Model Predictive Control. In Lecture Notes in Computer Science (pp. 147–159). Springer International Publishing. [https://doi.org/10.1007/978-3-319-96983-1\\_11](https://doi.org/10.1007/978-3-319-96983-1_11)
73. Calheiros, R. N., Toosi, A. N., Vecchiola, C., & Buyya, R. (2012). A coordinator for scaling elastic applications across multiple clouds. *Future Generation Computer Systems*, 28(8), 1350–1362. <https://doi.org/10.1016/j.future.2012.03.010>
74. Rodriguez, M. A., & Buyya, R. (2018). Container-based Cluster Orchestration Systems: A Taxonomy and Future Directions (Version 2). arXiv. <https://doi.org/10.48550/ARXIV.1807.06193>
75. Al-Haidari, F., Sqalli, M. H., & Salah, K. (2013). Impact of CPU utilization thresholds and scaling size on autoscaling cloud resources. In 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (Vol. 2, pp. 256–261). IEEE
76. Masdari, M., Nabavi, S. S., & Ahmadi, V. (2016). An overview of virtual machine placement schemes in cloud computing. In *Journal of Network and Computer Applications* (Vol. 66, pp. 106–127). Elsevier BV. <https://doi.org/10.1016/j.jnca.2016.01.011>
77. Sotiriadis, S., Bessis, N., & Buyya, R. (2018). Self managed virtual machine scheduling in Cloud systems. In *Information Sciences* (Vols. 433–434, pp. 381–400). Elsevier BV. <https://doi.org/10.1016/j.ins.2017.07.006>
78. Halushko, D., Rolik, O., & Samotyy, V. (2017) A Load Balancing Mechanism Based on Fuzzy Nonparametric Analysis of QoS Parameters. In *Proc. CLOUD*

COMPUTING 2017: The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization. Athens, Greece, IARIA.

79. Jafarnejad Ghomi, E., Masoud Rahmani, A., & Nasih Qader, N. (2017). Load-balancing algorithms in cloud computing: A survey. In *Journal of Network and Computer Applications* (Vol. 88, pp. 50–71). Elsevier BV. <https://doi.org/10.1016/j.jnca.2017.04.007>
80. Afzal, S., & Kavitha, G. (2019). Load balancing in cloud computing – A hierarchical taxonomical classification. In *Journal of Cloud Computing* (Vol. 8, Issue 1). Springer Science and Business Media LLC. <https://doi.org/10.1186/s13677-019-0146-7>
81. Chauhan, N., Kaur, N., Saini, K. S., Verma, S., Alabdulatif, A., Khurma, R. A., Garcia-Arenas, M., & Castillo, P. A. (2024). A Systematic Literature Review on Task Allocation and Performance Management Techniques in Cloud Data Center (Version 1). arXiv. <https://doi.org/10.48550/ARXIV.2402.13135>
82. Kubernetes Documentation. Kubernetes. <https://kubernetes.io/docs>
83. Prometheus Documentation. Prometheus. <https://prometheus.io/docs>

## ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ В НАВЧАЛЬНИЙ ПРОЦЕС

Декан факультету інформатики  
та обчислювальної техніки  
КПІ ім. Сікорського

 Ярослав КОРНАГА  
«03» 01 2016 року

## АКТ

про впровадження в навчальний процес кафедри Інформаційних систем та технологій факультету Інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» результатів дисертаційної роботи Омельченка Віталія Вікторовича «Інформаційна технологія управління обчислювальними ресурсами в Kubernetes кластері», поданої на здобуття наукового ступеня доктора філософії.

Ми, що нижче підписалися: заступник завідувача кафедри Інформаційних систем та технологій, д.т.н. проф. Теленик С. Ф., вчений секретар кафедри, к.ф.-м.н., доц. Гавриленко О. В. підтверджуємо, що результати дисертаційної роботи Омельченка В. В. були включені в матеріали навчально-методичного забезпечення курсів «Інфраструктура інформаційних технологій» та «Методи та засоби управління інфраструктурою інформаційних технологій», а саме:

- метод проактивного масштабування для роботи з навантаженнями, що містять комплексні сезонності та тенденції, що дозволяє забезпечити встановлений рівень якості послуг при мінімально можливому використанні обчислювальних ресурсів;
- гібридний метод масштабування, що поєднує реактивний і проактивний компоненти, в якому передбачена оперативна передача управління у разі відсутності історичних даних, недостатньої точності отриманих прогнозів або некоректної роботи проактивного компонента;
- метод поєднання горизонтального і вертикального компонентів масштабування з використанням модуля координації, що дозволяє покращити ефективність використання обчислювальних ресурсів при горизонтальному масштабуванні.


Впровадження результатів дисертаційної роботи Омельченко В. В. в навчальний процес дозволило підвищити якість підготовки студентів освітнього рівня «Бакалавр» 126 «Інформаційні системи та технології» на кафедрі Інформаційних систем та технологій КПІ ім. Ігоря Сікорського.

Заступник завідувача кафедри  
інформаційних систем та технологій,  
д.т.н., проф.



Сергій ТЕЛЕНИК

Вчений секретар кафедри  
інформаційних систем та технологій,  
к.ф.-м.н., доц.



Олена ГАВРИЛЕНКО

## ДОДАТОК Б. ЛІСТИНГ КОДУ КОМБІНОВАНОГО МЕТОДУ ПРОГНОЗУВАННЯ

```
def predict_pattern_matching(df, prediction_period=1440 * 2):
    def predict_pattern_matching(df, prediction_periods):
        l = len(df)
        split_index = l - LAST_PERIODS_TO_PREDICT
        train_df, to_predict_df = df.iloc[:split_index],
df.iloc[split_index:]
        train = train_df['y'].values
        to_predict = to_predict_df['y'].values
        distance_matrix = stumpy.mass(to_predict, train)
        idx = np.argmin(distance_matrix)
        prediction = df['y'].values[idx:idx +
prediction_periods]

        future_dates = pd.date_range(start=df['ds'].iloc[-1] +
pd.Timedelta(minutes=1), periods=prediction_periods,
                                freq='min')
        print(len(future_dates), len(prediction))
        predicted_df = pd.DataFrame({'ds': future_dates, 'y':
prediction})
        return predicted_df

    def find_best_match(actual, predicted, l):
        def f(a, b):
            diff = np.mean(
                np.abs((a.reset_index(drop=True).set_index('ds')
- b.reset_index(drop=True).set_index('ds'))))
            return diff
        def step(x):
            from datetime import timedelta
            return x - timedelta(minutes=1)
        a = actual.copy()
        b = predicted.copy()

        min_accuracy = float('inf')
        min_offset = -1
        for offset in range(0, l, 1):
            b['ds'] = b['ds'].apply(step)
            accuracy = f(a, b)
            if not math.isnan(accuracy) and accuracy <
min_accuracy:
                min_accuracy = accuracy
                min_offset = offset

        result = predicted.copy()
        result['ds'] = result['ds'].apply(lambda x: x -
timedelta(minutes=min_offset))
        k = 1
        result['y'] /= k
        return result
```

```

    predicted = predict_pattern_matching(df, prediction_period)
    match = find_best_match(df.iloc[-LAST_PERIODS_TO_PREDICT:],
predicted, LAST_PERIODS_TO_PREDICT)
    return match

def predict_prophet(df, prediction_period):
    import prophet as p
    prophet = p.Prophet(daily_seasonality=True,
weekly_seasonality=True)
    prophet.fit(df)
    future = prophet.make_future_dataframe(periods=1440 * 1,
freq='min')
    forecast = prophet.predict(future)
    return forecast[['ds', 'yhat']].rename(columns={"ds": "ds",
"yhat": "y"})

def combine(q, a, b, l=200):
    def mean_absolute_percentage_error(y_true, y_pred):
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

    def optimize_weights(a, b, q):
        a = a.iloc[:l].reset_index(drop=True)
        b = b.iloc[:l].reset_index(drop=True)
        q = q.iloc[:l].reset_index(drop=True)
        a_values = a['y'].values
        b_values = b['y'].values
        q_values = q['y'].values
        def objective(weights):
            w1, w2 = weights
            combined_predictions = w1 * a_values + w2 * b_values
            return mean_absolute_percentage_error(q_values,
combined_predictions)

        constraints = ({'type': 'eq', 'fun': lambda weights: 1 -
sum(weights)})
        bounds = [(0, 1), (0, 1)]
        initial_weights = [0.5, 0.5]
        result = minimize(objective, initial_weights,
bounds=bounds, constraints=constraints, method='SLSQP')

        if result.success:
            optimized_weights = result.x
        else:
            raise ValueError("Optimization failed")

    return optimized_weights

weights = optimize_weights(a, b, q)
print(f"Optimized weights: w1 = {weights[0]:.4f}, w2 =
{weights[1]:.4f}")
return weights

```

## ДОДАТОК В. ЛІСТИНГ КОДУ ПРОГРАМНОГО МОДУЛЮ ПРОАКТИВНОГО МАСШТАБУВАННЯ

```
def get_future_replicas(prediction: dict, env: Env) -> int:
    current = datetime.today().replace(microsecond=0) +
timedelta(
        seconds=1
    )
    future = datetime.today().replace(microsecond=0) +
timedelta(
        seconds=env.app_latency
    )
    two_steps_ahead_future =
datetime.today().replace(microsecond=0) + timedelta(
        seconds=env.app_latency * 2
    )
    current_replicas = int(math.ceil(prediction[current] /
env.cpu_per_pod))
    replicas = int(math.ceil(prediction[future] /
env.cpu_per_pod))
    two_steps_ahead_replicas =
int(math.ceil(prediction[two_steps_ahead_future] /
env.cpu_per_pod))
    print("replicas", current_replicas, replicas,
two_steps_ahead_replicas)

    return min(max(current_replicas, replicas,
two_steps_ahead_replicas), MAX_PODS)

if __name__ == "__main__":
    env = read_env()
    kubernetes.config.load_config()

    kube_client = kubernetes.client.CoreV1Api()
    prom_client = get_client()
    prediction = None
    model_cache = datetime.now() -
timedelta(seconds=(MODEL_CACHE_TIMEOUT + 1))
    log.info("env %s:", env)

    log.info("HPA is enabled")
    enable_hpa(env.deployment, env.namespace)

    time_started = time.time()

    while True:
        now = datetime.now()

        if model_cache + timedelta(seconds=MODEL_CACHE_TIMEOUT)
<= now:
            metrics = read_prometheus_metrics(
```



```

        prom_client, env.deployment, env.namespace
    )
    prediction, accuracy = predict(metrics,
SEASONALITY_DAYS)
    model_cache = now
    log.info(f"Got prediction accuracy={int(accuracy *
100)}%")

    if accuracy < ACCURACY_THRESHOLD:
        disable_hpa(env.deployment, env.namespace)
        continue
    else:
        enable_hpa(env.deployment, env.namespace)

    assert prediction
    future_replicas = get_future_replicas(prediction, env)

    with kubernetes.client.ApiClient() as api_client:
        api_instance =
kubernetes.client.AppsV1Api(api_client)
        api_response =
api_instance.patch_namespaced_deployment_scale(
            env.deployment, env.namespace, {"spec":
{"replicas": future_replicas}}
        )

    log.info(f"Set replicas to {future_replicas}")

    time.sleep(env.interval)

```

## ДОДАТОК Г. ЛІСТИНГ КОДУ ПРОГРАМНОГО МОДУЛЮ ГІБРИДНОГО МАСШТАБУВАННЯ

```
def strategy_cpu_reactive(
    history: t.List[t.Tuple[int, float]],
    target_percentile: int = 95,
    max_data_length: int = DAY_IN_SECONDS * 7,
) -> float:
    target_history = history[::-1][max_data_length:]
    sorted_history = list(sorted(target_history, key=lambda
time_n_value: time_n_value[1]))
    percentile_index = len(sorted_history) // 100 *
target_percentile
    _, recommended_cpu_value = sorted_history[percentile_index]
    return recommended_cpu_value

def strategy_mem_reactive(
    history: t.List[t.Tuple[int, float]],
    overhead_factor: float = 0.1,
    max_data_length: int = DAY_IN_SECONDS * 7,
):
    target_history = history[::-1][max_data_length:]
    return max(time_n_value[1] for time_n_value in
target_history) * (1 + overhead_factor)
```