Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Програмні засоби моніторингу споживання ресурсів комп'ютерної системи

(тема)

Виконав:

здобувач 4 року навчання,

групи КІУКІиз-21-1

Вугар ТАЛІБОВ
(власне ім’я, прізвище)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Дмитро РОСІНСЬКИЙ
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Талібов Вугар Васіф огли
(прізвище, ім'я, по батькові)

1. Тема роботи Програмні засоби моніторингу споживання ресурсів комп'ютерної системи

затверджена наказом по університету від "05" травня 2025 р. № 73 Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) апаратно-програмна платформа: персональний комп'ютер IBM PC AT під керуванням ОС Linux; 2) моніторингу підлягають ресурси центрального процесора, оперативної пам'яті, дискових приводів.

4. Перелік питань, що потрібно опрацювати у роботі
Методи та підходи до моніторингу ресурсів комп'ютерних систем
Огляд існуючих програмних засобів моніторингу ресурсів
Вибір технологій та програмних засобів
Архітектура системи моніторингу
Алгоритми збору та аналізу моніторингових даних
Реалізація програмних засобів моніторингу
Тестування та аналіз результатів роботи програмних засобів моніторингу

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 13 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	06.05.25-09.05.25	
2	Вибір технологій та програмних засобів	10.05.25-13.05.25	
3	Розробка алгоритмічного забезпечення	14.05.25-17.05.25	
4	Розробка програмних модулів	19.05.25-28.05.25	
5	Тестування програмних засобів	29.05.25-05.06.25	
6	Оформлення матеріалів кваліфікаційної роботи	06.06.25-09.06.25	
7	Подання кваліфікаційної роботи керівникові та	10.06.25-11.06.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	12.06.25-13.06.25	
	вання		

Дата видачі завдання " 05 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Дмитро РОСІНСЬКИЙ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 59 с., 5 рис., 5 табл., 1 дод., 24 джерела.

ІНТЕРФЕЙС, КОМП'ЮТЕРНА СИСТЕМА, МОНІТОРИНГ, ПРОДУКТИВНІСТЬ, СПОЖИВАННЯ РЕСУРСУ.

Метою роботи є аналіз існуючих програмних засобів моніторингу ресурсів комп'ютерних систем, розробка та реалізація власного рішення, що дозволить забезпечити ефективний контроль та аналіз споживання ресурсів.

Для досягнення поставленої мети вирішено такі завдання: аналіз сучасних методів та підходів до моніторингу ресурсів; порівняльний огляд існуючих програмних рішень; розробка архітектури та обґрунтування вибору інструментів для реалізації системи моніторингу; реалізація систему моніторингу; тестування та оцінити ефективність створеної системи.

Об'єктом дослідження виступають процеси моніторингу споживання ресурсів комп'ютерної системи. Предметом дослідження є програмні засоби та методи аналізу та контролю ресурсів комп’ютерних систем. У процесі виконання роботи використано методи порівняльного аналізу, проєктування програмного забезпечення, експериментальні дослідження та оцінювання ефективності.

ABSTRACT

Bachelor’s thesis: 59 pages, 5 figures, 5 tables, 1 appendix, 24 sources.

COMPUTER SYSTEM, INTERFACE, MONITORING, PERFORMANCE, RESOURCE CONSUMPTION.

The aim of this work is to analyze existing software tools for monitoring computer system resources, as well as to design and implement a custom solution that enables efficient control and analysis of resource consumption.

To achieve this aim, the following objectives have been accomplished: analysis of modern methods and approaches to resource monitoring; comparative review of existing software solutions; design of the architecture and justification of the selected tools for implementing the monitoring system; implementation of the monitoring system; testing and evaluation of the effectiveness of the developed system.

The object of the research is the processes of monitoring the resource consumption of a computer system. The subject of the research is the software tools and methods for analyzing and controlling computer system resources. The study employs methods of comparative analysis, software design, experimental research, and effectiveness evaluation.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ІТ – інформаційні технології

ОС – операційна система

ПЗ – програмне забезпечення

APM – моніторинг продуктивності застосунків (англ., Application Performance Monitoring)

CPU – центральний процесор (англ., Central Processing Unit)

I/O – введення/виведення (англ., Input/Output)

ICMP – протокол керування повідомленнями Інтернету (англ., Internet Control Message Protocol)

IPMI – інтерфейс інтелектуального керування платформою (англ., Intelligent Platform Management Interface)

ML – машинне навчання (англ., Machine Learning)

RAM – оперативна пам’ять (англ., Random Access Memory)

SNMP – простий протокол керування мережею (англ., Simple Network Management Protocol)

TSDB – база даних часових рядів (англ., Time Series Database)

WMI – інструментарій керування Windows (англ., Windows Management Instrumentation)

ВСТУП

Зі стрімким зростанням масштабів комп'ютеризації та впровадженням інформаційних технологій у всі сфери людської діяльності виникає необхідність у забезпеченні ефективної роботи комп’ютерних систем. Однією з ключових умов цієї ефективності є раціональне використання обчислювальних ресурсів, до яких належать процесорний час, оперативна пам'ять, дисковий простір та мережеві канали. У зв’язку з цим питання моніторингу споживання ресурсів набуває особливої актуальності, оскільки дозволяє оперативно реагувати на перевантаження системи, запобігати відмовам і забезпечувати стабільність її роботи [1].

Завдяки використанню сучасних програмних засобів моніторингу можна своєчасно виявляти та прогнозувати виникнення критичних ситуацій, що впливає на безпеку даних та якість сервісу. Це особливо важливо в умовах високонавантажених інформаційних систем, таких як сервери великих підприємств, хмарні сервіси або центри обробки даних [2]. Водночас існуючі рішення мають свої переваги та недоліки, а вибір конкретного програмного засобу залежить від специфіки завдань, умов експлуатації системи, а також вимог до точності та частоти збору моніторингових даних [3].

Метою даної бакалаврської роботи є аналіз існуючих програмних засобів моніторингу ресурсів комп'ютерних систем, розробка та реалізація власного рішення, що дозволить забезпечити ефективний контроль та аналіз споживання ресурсів.

Для досягнення поставленої мети визначено такі завдання:

- провести аналіз сучасних методів та підходів до моніторингу ресурсів;
- виконати порівняльний огляд існуючих програмних рішень;
- розробити архітектуру та обґрунтувати вибір інструментів для

реалізації системи моніторингу;

- реалізувати систему моніторингу з використанням обраних технологій;
- провести тестування та оцінити ефективність створеної системи.

Об'єктом дослідження виступають процеси моніторингу споживання ресурсів комп'ютерної системи. Предметом дослідження є програмні засоби та методи аналізу та контролю ресурсів комп’ютерних систем.

У процесі виконання роботи було використано методи порівняльного аналізу, проєктування програмного забезпечення, експериментальні дослідження та оцінювання ефективності.

Структура роботи складається зі вступу, чотирьох основних розділів, висновків, списку використаних джерел та додатків.

1 ОСНОВИ МОНІТОРИНГУ СПОЖИВАННЯ РЕСУРСІВ КОМП'ЮТЕРНИХ СИСТЕМ

1.1 Поняття ресурсів комп’ютерної системи та необхідність їх моніторингу

Ефективне функціонування сучасних комп’ютерних систем безпосередньо пов’язане з оптимальним розподілом та контролем ресурсів, які забезпечують їхню працездатність. Під ресурсами комп’ютерної системи зазвичай розуміють сукупність апаратних та програмних складових, таких як процесорний час, оперативна пам’ять, дисковий простір і ресурси мережі, які необхідні для виконання завдань користувача та системних процесів [1].

Процесорний час є одним із найважливіших ресурсів, оскільки саме він визначає швидкість обробки інформації, а також здатність системи ефективно виконувати одночасно кілька завдань. Недостатній або нераціонально розподілений процесорний час може призводити до суттєвого зниження продуктивності комп’ютерної системи, її уповільнення або навіть до повного збою у роботі програмних засобів, особливо у високонавантажених середовищах [2].

Оперативна пам’ять є ще одним критично важливим ресурсом, оскільки саме вона забезпечує тимчасове зберігання даних та програмних кодів, які безпосередньо використовуються під час роботи. За умов недостатнього обсягу або неефективного використання оперативної пам’яті може відбуватися активний обмін даними із постійними носіями (дисками), що значно сповільнює роботу системи та знижує її загальну продуктивність [4].

Дисковий простір, у свою чергу, є ресурсом, що визначає можливість зберігання та швидкість доступу до великих обсягів інформації. Недостатня кількість вільного місця на дисках може призводити до зупинки критично

важливих процесів, втрати даних або неможливості їх своєчасного збереження, що у підсумку негативно впливає на надійність та функціональність системи в цілому [2].

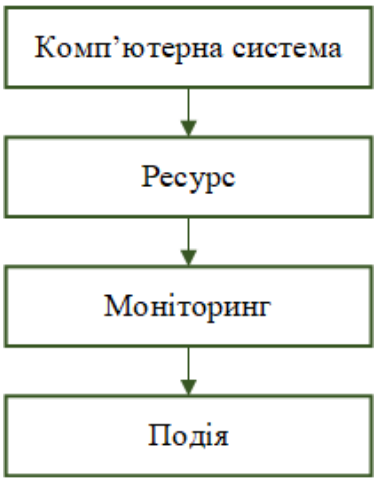

Рисунок 1.1 – Концепція моніторингу комп’ютерної системи

Окрім цього, важливу роль відіграють мережеві ресурси, що забезпечують передачу даних між різними комп’ютерними системами та вузлами. Їх нераціональне використання або недостатня пропускна здатність можуть призводити до втрати пакетів даних, затримок у роботі програмних додатків та погіршення загального рівня обслуговування користувачів, особливо у розподілених системах та хмарних середовищах [5].

В умовах активного використання комп'ютерних систем, особливо при високій динаміці навантажень та значній кількості паралельних процесів, виникає гостра потреба у постійному та детальному моніторингу використання ресурсів. Без належного контролю адміністратори не мають змоги оперативно реагувати на перевантаження окремих компонентів, що в результаті може призводити до зниження продуктивності, а в гірших випадках – до часткової або повної зупинки роботи інформаційних систем. Відсутність системного моніторингу також ускладнює процес аналізу та прогнозування можливих збоїв у роботі комп’ютерних систем, а отже, ускладнює й ухвалення своєчасних управлінських рішень [6].

Важливість моніторингу ресурсів зумовлюється також необхідністю забезпечення інформаційної безпеки та надійності систем. Регулярний аналіз ресурсів дозволяє вчасно виявити потенційні загрози, наприклад, надмірне навантаження через вірусні атаки, витоки пам’яті чи надмірне використання дискових ресурсів через неконтрольовані процеси, що може спричинити суттєві фінансові та репутаційні втрати для організацій [7].

Таким чином, моніторинг ресурсів комп’ютерних систем є критично важливою функцією, яка забезпечує оптимальний баланс продуктивності, безпеки та стабільності роботи інформаційних систем. Саме тому вивчення та застосування ефективних програмних засобів для реалізації моніторингу є надзвичайно актуальним і перспективним напрямком сучасних досліджень у галузі інформаційних технологій. Рисунок 1.2 дозволяє чітко візуалізувати поняття та взаємозв'язки, викладені у підрозділі 1.1.

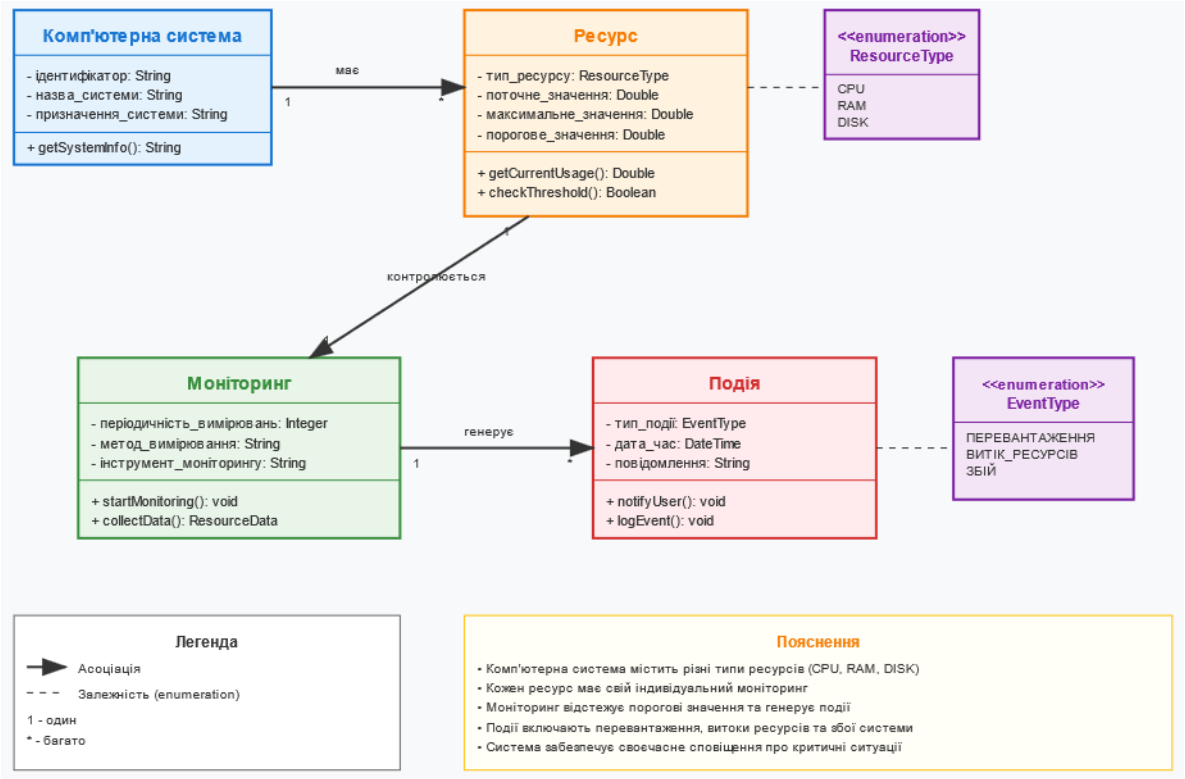

Рисунок 1.2 – Ресурси комп'ютерної системи та необхідність їх моніторингу

1.2 Методи та підходи до моніторингу ресурсів комп’ютерних систем

Моніторинг комп’ютерних ресурсів – це процес безперервного збору, аналізу та збереження інформації про стан обчислювальної системи для своєчасного виявлення перевантажень, збоїв та відхилень від нормального функціонування. Для реалізації цієї задачі застосовуються різні методи і підходи, які відрізняються за архітектурою, ступенем втручання в систему, гнучкістю налаштувань та продуктивністю.

Один із базових поділів методів ґрунтується на розмежуванні активного та пасивного моніторингу. Активний моніторинг передбачає ініціативне звернення до ресурсів або сервісів з метою перевірки їхньої доступності, навантаження або продуктивності. При цьому в систему надсилаються спеціальні запити, які можуть спричиняти додаткове навантаження. Цей підхід широко використовується у мережевому моніторингу, наприклад, за допомогою протоколу ICMP (ping), SNMP-запитів або HTTP health-checks [8]. Пасивний моніторинг, навпаки, базується на аналізі вже наявних логів, метрик і системних повідомлень без прямого втручання в роботу об’єктів спостереження. Такий підхід дозволяє знизити навантаження, але водночас може бути менш оперативним у виявленні проблем [9].

З погляду архітектурної реалізації, розрізняють агентний і безагентний моніторинг. Агентні системи використовують спеціальні програми (агенти), які встановлюються на вузлах, що моніторяться, і передають дані на центральний сервер або в систему обробки. Такий підхід дозволяє отримувати деталізовану інформацію, у тому числі про внутрішні метрики ядра, системних процесів, апаратного забезпечення, наприклад за допомогою Node Exporter (Prometheus), collectd або Zabbix Agent [10]. Безагентні системи взаємодіють із цільовими системами через загальнодоступні інтерфейси, такі як API, SNMP, WMI або лог-файли. Вони менш інвазивні, але можуть мати обмежену функціональність [11].

Сучасні підходи до моніторингу активно використовують пулінгові (pull) та пушингові (push) моделі збору даних. У пулінговій моделі центральна система опитує клієнтські вузли з певною періодичністю, тоді як у пушинговій – клієнти самостійно надсилають інформацію про свій стан на сервер. Наприклад, Prometheus застосовує pull-модель, що дозволяє краще контролювати навантаження, тоді як Telegraf (у складі TICK-стека) підтримує push-архітектуру, зручно інтегруючись із різними типами систем [12].

Окрему нішу займають гібридні системи моніторингу, які поєднують активні та пасивні методи, а також підтримують як агентні, так і безагентні механізми. Це дозволяє досягти балансу між детальністю моніторингу та мінімізацією впливу на продуктивність. Наприклад, Zabbix підтримує комбінований підхід, дозволяючи як агентне опитування, так і SNMP, IPMI, лог-моніторинг, та HTTP-чеки в єдиній платформі [13].

Із розвитком хмарних і контейнеризованих технологій виникла потреба в нових методах, які включають контейнер-орієнтований моніторинг (наприклад, cAdvisor для Docker) і моніторинг в хмарних середовищах (AWS CloudWatch, Azure Monitor, Google Cloud Operations Suite). Ці системи дозволяють інтегрувати моніторинг із CI/CD-пайплайнами, масштабувати збір метрик у реальному часі та автоматизувати реагування за допомогою тригерів і правил [14].

Загалом, вибір підходу до моніторингу залежить від багатьох факторів: рівня деталізації, продуктивності, масштабованості, безпеки та зручності адміністрування. Ефективна система моніторингу повинна забезпечувати високу точність виявлення аномалій, гнучкість у налаштуванні, а також інтеграцію з інструментами візуалізації, аналітики та оповіщення.

Значною сучасною тенденцією у розвитку моніторингу є інтеграція з технологіями машинного навчання (ML), які дозволяють не лише фіксувати поточні стани системи, але й здійснювати прогнозування на основі історичних даних. Зокрема, використовуються методи класифікації, регресії

та виявлення аномалій для автоматичного виявлення нетипової поведінки системних ресурсів, що може свідчити про потенційні атаки або відмови апаратного забезпечення [15]. Такі рішення особливо цінні в контексті DevOps-практик і підходів до самообслуговування інфраструктури (self-healing systems).

Іншим актуальним напрямом є кореляція моніторингових даних із журналами подій, системами керування інцидентами (наприклад, ELK Stack, Splunk, Graylog) та APM-платформами (Application Performance Monitoring), що забезпечує не лише спостереження, а й розуміння причинно-наслідкових зв’язків між ресурсним навантаженням, подіями в системі та поведінкою користувачів [16].

Особливу увагу в сучасних підходах приділяють візуалізації моніторингових даних, адже якісно побудовані дашборди (наприклад, у Grafana або Kibana) дозволяють операторам та адміністраторам оперативно і точно оцінити поточний стан системи, порівняти історичні тренди, визначити аномальні піки споживання ресурсів або підтвердити нормальний стан системи. Добре організована візуалізація сприяє зниженню часу реагування на інциденти та пришвидшує ухвалення рішень [17].

Важливо також зазначити, що підходи до моніторингу значною мірою залежать від масштабів та призначення системи, що моніториться. Наприклад, для малих локальних систем може бути достатньо простих інструментів на основі cron-скриптів або вбудованих системних утиліт (top, htop, iostat, netstat), тоді як для кластерних середовищ або систем із мікросервісною архітектурою критично необхідні масштабовані рішення з розподіленою архітектурою, можливістю горизонтального масштабування та збирання великої кількості метрик у режимі реального часу [18].

Насамкінець, ефективний моніторинг повинен не лише відображати дані, але й інтегруватися з системами оповіщення та автоматичного реагування. Інтеграція з сервісами Telegram, Slack, Email або навіть системами автоматичного масштабування дозволяє в режимі реального часу

не лише фіксувати проблему, а й оперативно усувати її наслідки, що є ключовим фактором у забезпеченні надійності й безперервності функціонування сучасних інформаційних систем.

Таким чином, методи та підходи до моніторингу ресурсів постійно еволюціонують, відповідаючи на виклики масштабованості, безпеки, продуктивності та гнучкості. Вибір правильного підходу має базуватися на глибокому розумінні потреб конкретної системи, ресурсних обмежень і вимог до її безперебійного функціонування.

1.3 Огляд існуючих програмних засобів моніторингу ресурсів

На сучасному етапі розвитку інформаційних технологій існує велика кількість інструментів для моніторингу ресурсів комп’ютерних систем, які відрізняються архітектурою, функціональними можливостями, підтримуваними платформами та рівнем інтеграції з іншими системами. Доцільний вибір таких засобів визначається цілями моніторингу, масштабами інфраструктури, вимогами до візуалізації даних і частоти оновлення метрик.

Першу категорію програмних засобів становлять вбудовані інструменти операційних систем. У середовищі Windows користувачам доступні диспетчер завдань (Task Manager), засіб моніторингу ресурсів (Resource Monitor) та утиліти perfmon і Get-Counter у PowerShell, які забезпечують базовий рівень спостереження за завантаженням процесора, оперативної пам’яті, дискової підсистеми та мережі. В середовищі Unix-подібних ОС (Linux, FreeBSD) типовими є утиліти top, htop, iotop, vmstat, netstat, nmon та інші. Вони дозволяють у режимі реального часу відслідковувати динаміку споживання ресурсів, але мають обмежені можливості для збереження історичних даних або інтеграції з візуалізаційними системами [2].

Серед агентно-орієнтованих систем, які підтримують масштабоване

середовище, варто виділити Zabbix – потужну систему з відкритим кодом, що забезпечує комплексний моніторинг серверів, мереж, віртуалізаційних середовищ, баз даних та прикладного ПЗ. Zabbix підтримує SNMP, IPMI, WMI, лог-моніторинг, автоматичне виявлення вузлів, тригери з умовами, та інтеграцію з оповіщеннями. Перевагою системи є централізована архітектура, яка добре масштабується, однак її конфігурація може потребувати значних зусиль на етапі налаштування [13].

Іншим популярним рішенням є Prometheus – система моніторингу з відкритим кодом, орієнтована на збір числових метрик за допомогою pull-механізму. Prometheus використовує власний мову запитів PromQL, підтримує автоматичне виявлення цілей моніторингу, експортерів метрик (наприклад, node_exporter) та добре інтегрується з платформою Grafana для візуалізації. Prometheus є стандартом де-факто у середовищі Kubernetes, завдяки гнучкості, масштабованості та можливості горизонтального шардінгу [19].

Nagios є класичним рішенням для моніторингу систем і сервісів, яке підтримує плагінову архітектуру та гнучку систему оповіщень. Його перевагами є висока надійність і простота розгортання в невеликих інфраструктурах. Проте через відсутність вбудованої системи зберігання історичних даних та обмежену гнучкість у налаштуванні сучасних візуалізацій Nagios поступається більш динамічним рішенням, як-от Prometheus або Zabbix [20].

Для візуалізації моніторингових даних найпоширенішим рішенням є Grafana, яка забезпечує інтерактивні дашборди, підтримку численних джерел даних (Prometheus, InfluxDB, Elasticsearch, Loki, MySQL та ін.), а також можливість створення кастомізованих панелей спостереження із динамічними параметрами. Grafana дозволяє легко виявляти аномалії, відстежувати історичні тренди та інтегрувати дані з APM-інструментами [17].

У середовищі контейнеризації широкого розповсюдження набув

cAdvisor – утиліта від Google, яка дозволяє моніторити використання ресурсів окремими контейнерами Docker. cAdvisor збирає метрики CPU, пам’яті, I/O та мережі, що є критично важливим у кластерних середовищах на базі Kubernetes [21].

Окрім того, у корпоративному середовищі активно використовуються комерційні рішення, такі як Datadog, New Relic, Dynatrace, які забезпечують глибоку інтеграцію з хмарними платформами (AWS, Azure, GCP), можливість відстеження стану мікросервісів, автоматичну побудову залежностей та кореляцію інцидентів. Основним недоліком таких систем є висока вартість і залежність від хмарної інфраструктури провайдера [22].

Для порівняльного аналізу основних інструментів моніторингу таблиця 1.1 узагальнює ключові характеристики популярних засобів.

Таблиця 1.1 – Порівняння популярних інструментів моніторингу

Засіб	Тип моніторингу	Візуалізація	Зберігання історії	Архітектура	Ліцензія
Zabbix	Агентний/безагентний	Вбудована	Так	Централіз.	Відкрита
Prometheus	Безагентний (pull)	Через Grafana	Так	Розподілена	Відкрита
Nagios	Агентний	Обмежена	Ні (без доповнень)	Централіз.	Відкрита
Grafana	–	Потужна	Через бекенди	–	Відкрита
cAdvisor	Безагентний	Обмежена	Так	Контейнерна	Відкрита
Datadog	Агентний/хмарний	Потужна	Так	Хмарна	Комерційна

Таким чином, ринок програмних засобів моніторингу є надзвичайно різноманітним і динамічним. Оптимальний вибір інструментів повинен базуватися на балансі між функціональністю, масштабованістю, простотою інтеграції та вартістю володіння.

2 ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ СПОЖИВАННЯ РЕСУРСІВ

2.1 Вибір технологій та програмних засобів

Під час проєктування системи моніторингу споживання ресурсів комп’ютерної системи важливим кроком є вибір відповідних платформ, технологій і інструментів, які дозволяють забезпечити надійність, масштабованість та зручність аналізу даних. У цьому дослідженні як базову обчислювальну платформу було обрано операційну систему Linux (дистрибутив Ubuntu Server), що обумовлено її відкритим кодом, широкою підтримкою спільноти, стабільністю в роботі серверних середовищ і гнучкістю у налаштуванні системних служб.

2.1.1 Ядро моніторингу

У ролі ядра системи моніторингу обрано Prometheus – високопродуктивну систему для збирання і обробки метрик у вигляді часових рядів. Prometheus має вбудовану Time Series Database (TSDB) та використовує pull-модель для отримання даних із клієнтів, що дає змогу контролювати частоту опитування та зменшити навантаження на систему.

Переваги Prometheus:

- ефективна робота з часовими рядами;
- мова запитів PromQL для побудови аналітичних виразів;
- підтримка масштабованих архітектур;
- легка інтеграція з експортерами та іншими сервісами;
- відкрите програмне забезпечення з активною спільнотою.

Prometheus було створено у 2012 році компанією SoundCloud і з того часу стало одним із основних компонентів екосистеми Cloud Native

Computing Foundation (CNCF) [19].

2.1.2 Засіб збору метрик

Для безпосереднього збору метрик із серверної операційної системи (у даному випадку Linux) використовується Node Exporter – офіційний експортер для Prometheus [23]. Цей інструмент запускається як фоновий процес і автоматично надає набір метрик за такими категоріями, як:

- завантаження процесора (CPU load);
- використання оперативної пам’яті (RAM usage);
- активність файлових систем (disk I/O);
- мережевий трафік (network throughput);
- стан swap-пам’яті, тощо.

Node Exporter не потребує складної конфігурації, є дуже легким та не створює суттєвого навантаження на систему. Він підтримує понад 30 колекторів метрик, які можуть увімкнутись або вимкнутись за потребою.

Основні переваги:

- оптимізований для Linux-систем;
- повністю сумісний із Prometheus;
- мінімальне споживання ресурсів;
- можливість масштабованого розгортання у кластерах.

2.1.3 Засоби візуалізації

Для візуалізації моніторингових даних обрано Grafana [17] – одну з найпотужніших open source платформ для побудови аналітичних дашбордів. Grafana забезпечує гнучку систему панелей, де користувач може відображати метрики в реальному часі, формувати звіти, виявляти аномалії за історичними даними та налаштовувати системи сповіщення.

Grafana підтримує численні джерела даних, включно з Prometheus,

InfluxDB, Elasticsearch, PostgreSQL, MySQL та багатьма іншими. У контексті цієї системи вона використовується як фронтенд до бази метрик Prometheus.

Переваги Grafana:

- гнучкі та динамічні візуалізації;
- підтримка шаблонів та змінних;
- побудова кастомізованих дашбордів;
- можливість створення тригерів та інтеграція з системами сповіщення (Slack, Email, Telegram, Webhook).

Для збереження історичних даних використовується вбудована TSDB-база Prometheus, яка зберігає часові ряди з ефективною компресією. За потреби систему можна доповнити довготривалим сховищем (наприклад, VictoriaMetrics або Thanos), але для цілей цієї роботи цього не передбачено.

Передача даних у системі здійснюється за допомогою HTTP-запитів у форматі text/plain, відповідно до специфікації Prometheus exposition format. Такий формат є легким, зручним для автоматичної обробки та не потребує зовнішніх бібліотек для реалізації.

Обрані технології зведені у таблицю 2.1 для зручного представлення.

Таблиця 2.1 – Вибір технологій для реалізації системи моніторингу

Компонент	Технологія / Засіб	Призначення
Операційна система	Ubuntu Server (Linux)	Серверне середовище
Збір метрик	Prometheus	База моніторингу та запитів
Експортер	Node Exporter	Збір системних метрик
Візуалізація	Grafana	Дашборди, графіки, сповіщення
Формат передачі	Prometheus exposition	Формат метрик (text/plain)
Протокол взаємодії	HTTP	Передача метрик і запитів

У підсумку, обраний стек технологій (Prometheus + Node Exporter + Grafana) забезпечує відкриту, легку у налаштуванні, масштабовану та ефективну систему моніторингу, яка є придатною як для локального

застосування, так і для розширення у майбутньому. Такий вибір є обґрунтованим з точки зору як технічних характеристик, так і вимог до простоти розгортання у навчальних і дослідницьких цілях.

2.2 Архітектура системи моніторингу

Обрана система моніторингу ресурсів комп'ютерної системи базується на архітектурі, що передбачає розділення функціональних компонентів на три основні рівні: рівень збору метрик, рівень зберігання та обробки, а також рівень візуалізації. Це забезпечує модульність, масштабованість і гнучкість у налаштуванні.

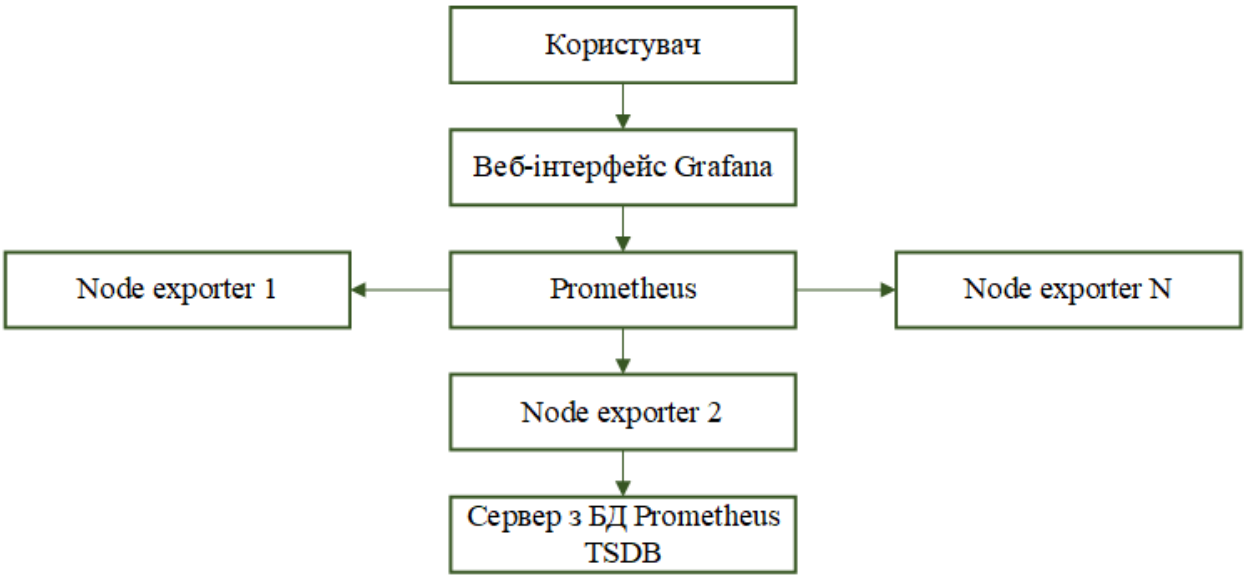

Рисунок 2.1 – Архітектура системи моніторингу

На рівні збору даних використовуються інстанси Node Exporter, встановлені на кожному вузлі, який підлягає моніторингу. Вони збирають дані про використання процесора, пам'яті, дисків, мережі тощо та надають їх через HTTP-інтерфейс у форматі, що відповідає специфікації Prometheus exposition format.

Сервер Prometheus [19] періодично звертається до всіх експортерів (pull-модель), отримує від них метрики, зберігає їх у власній time series базі

даних (TSDB) і надає API для аналітичних запитів. Prometheus також керує конфігурацією цільових вузлів, інтервалами збору даних і правилами оповіщення.

На рівні візуалізації працює Grafana [17], яка підключається до Prometheus як джерела даних і дозволяє будувати гнучкі, налаштовувані дашборди для візуального відображення стану ресурсів системи. Користувач отримує доступ до інформації через веб-інтерфейс Grafana з можливістю створення кастомних графіків, таблиць, алертів тощо.

Взаємодія компонентів:

- Node Exporter [23] працює як агент на кожному моніторинговому вузлі;
- Prometheus опитує всі інстанси Node Exporter за заданим графіком (наприклад, кожні 15 секунд);
- Grafana надсилає запити до Prometheus для отримання необхідних метрик;
- користувач взаємодіє з Grafana через браузер, створює візуалізації, задає фільтри й аналізує продуктивність системи;
- дані Prometheus зберігаються локально в TSDB та можуть бути інтегровані з зовнішніми сховищами (наприклад, Thanos, VictoriaMetrics).

Розроблена архітектура легко масштабується [12]:

- додавання нових вузлів полягає у встановленні Node Exporter та доданні його адреси до конфігураційного файлу Prometheus;
- горизонтальне масштабування Prometheus можливе через розбиття на декілька інстансів за зонами відповідальності або інтеграцію з Thanos;
- Grafana підтримує кілька джерел даних одночасно – можна об'єднувати інформацію з кількох Prometheus-серверів;
- передбачено можливість інтеграції з системами оповіщення (Slack, Email, Telegram) для оперативного інформування про критичні ситуації.

Таким чином, запропонована архітектура забезпечує не лише надійність і доступність даних про ресурси, а й зручність для

адміністрування та аналітики в реальному часі. Вона базується на відкритих стандартах, активно підтримується спільнотою та має потужний потенціал до масштабування.

2.3 Алгоритми збору та аналізу моніторингових даних

Структурна схема алгоритму та аналізу моніторингових даних наведена на рисунку 2.2.

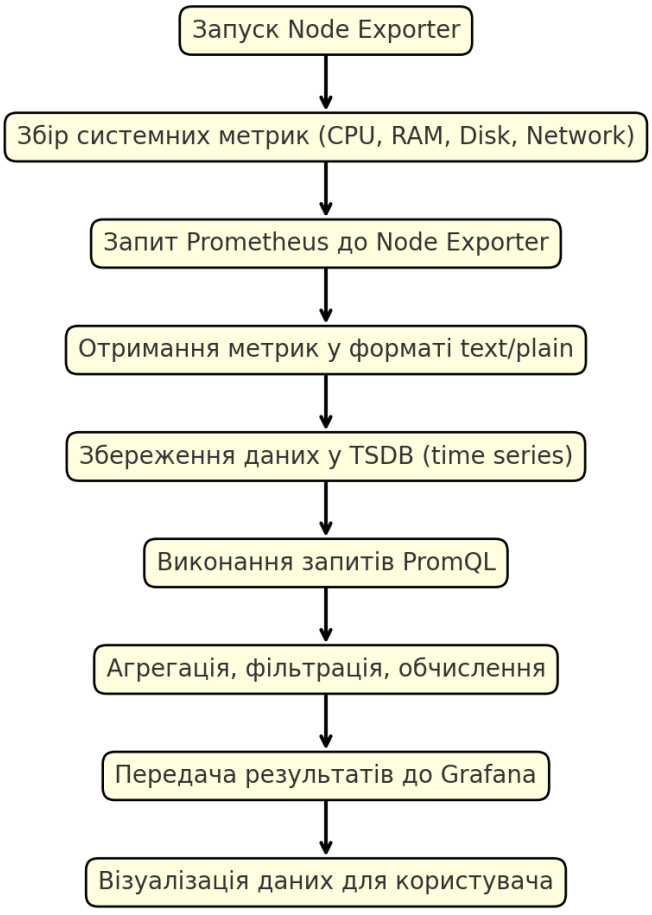

Рисунок 2.2 – Схема алгоритму збору та аналізу моніторингових даних

2.3.1 Збір метрик: взаємодія Node Exporter і Prometheus

Система збору моніторингових даних ґрунтується на періодичному опитуванні Prometheus усіх цільових вузлів, на яких запущено Node Exporter.

Кожен інстанс Node Exporter працює як HTTP-сервер, який публікує у відкритому доступі набір системних метрик у текстовому форматі відповідно до специфікації Prometheus exposition format.

Типовий HTTP-запит від Prometheus до Node Exporter має вигляд:

```
GET http://<host>:9100/metrics
```

У відповідь Prometheus отримує плаский список метрик у вигляді ключ-значення з можливими мітками (labels). Наприклад:

```
node_cpu_seconds_total{cpu="0",mode="user"} 25793.42
node_memory_MemAvailable_bytes 3.452681216e+09
node_filesystem_avail_bytes{fstype="ext4",mountpoint="/"}
1.203457024e+10
```

Основні категорії метрик Node Exporter:

- CPU (node_cpu_seconds_total) – час роботи процесора в різних режимах (user, system, idle);
- Memory (node_memory_*) – загальний обсяг, використана та доступна оперативна пам’ять;
- Disk (node_disk_, node_filesystem_) – використання файлових систем, I/O-операції;
- Network (node_network_*) – статистика передачі пакетів, помилки, затримки;
- Load (node_load1, node_load5, node_load15) – середнє навантаження на систему;
- Uptime (node_time_seconds, node_boot_time_seconds) – поточний час і момент останнього перезавантаження.

2.3.2 Частота збору даних

Частота опитування задається у конфігураційному файлі

prometheus.yml. Наприклад:

```
global:
  scrape_interval: 15s  # опитування кожні 15 секунд
scrape_configs:
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['localhost:9100']
```

Таким чином, кожні 15 секунд Prometheus отримує актуальні дані з усіх підключених вузлів та оновлює часові ряди у своїй вбудованій базі.

2.3.3 Модель зберігання даних у Prometheus

Prometheus зберігає метрики у форматі часових рядів (time series). Кожна метрика – це унікальна пара:

- ім’я метрики (наприклад, node_cpu_seconds_total);
- набір міток (labels) (наприклад, {cpu="0", mode="user"});
- часова позначка (timestamp);
- значення метрики (value).

Наприклад, один часовий ряд:

```
metric: node_memory_MemAvailable_bytes
labels: {instance="192.168.0.10:9100", job="node_exporter"}
timestamp: 1717190050
value: 3452681216
```

Prometheus автоматично агрегує ці дані та зберігає їх у власному сховищі – TSDB (Time Series Database), оптимізованому для високопродуктивного читання та стиснення. Стара інформація поступово видаляється відповідно до політики утримання (retention, за замовчуванням – 15 діб), проте її можна змінити або використовувати зовнішнє довготривале сховище (Thanos, Cortex, VictoriaMetrics).

2.3.4 Мова запитів PromQL: побудова аналітики

Для витягування, агрегації й аналізу метрик Prometheus використовує власну декларативну мову PromQL (Prometheus Query Language). PromQL дозволяє:

- фільтрувати метрики за мітками;
- обчислювати середні значення, максимуми, суму;
- виконувати операції над часовими рядами;
- будувати миттєві зрізи (instant vector) і часові діапазони (range vector).

Типові приклади запитів PromQL є такими.

1. Середнє завантаження CPU в користувацькому режимі (user mode):

```
rate(node_cpu_seconds_total{mode="user"}[5m])
```

Пояснення: розраховує швидкість приросту метрики node_cpu_seconds_total за останні 5 хвилин. Дозволяє побачити активність процесора.

2. Використання оперативної пам’яті:

```
1 - (node_memory_MemAvailable_bytes /
node_memory_MemTotal_bytes)
```

Пояснення: відношення доступної до загальної пам’яті. Результат – частка використаної пам’яті (наприклад, 0.73 = 73%).

3. Кількість доступного місця на кореневому диску:

```
node_filesystem_avail_bytes{mountpoint="/", fstype!="tmpfs"}
```

Пояснення: показує, скільки байтів вільно на кореневій файловій системі, виключаючи тимчасові файлові системи (tmpfs).

4. Мережевий вхідний трафік за останні 5 хвилин:

```
rate(node_network_receive_bytes_total[5m])
```

Пояснення: показує середню швидкість прийому даних у байтах за секунду.

5. Виведення лише з одного вузла:

```
rate(node_cpu_seconds_total{instance="192.168.0.10:9100",
mode="system"}[1m])
```

Пояснення: фільтрація за IP-адресою інстансу і режимом CPU – system.

2.3.5 Виявлення аномалій і використання алертів

Prometheus підтримує вбудований механізм генерації сповіщень (alerting rules). Вони визначаються в конфігурації й періодично перевіряються. Якщо умова істинна протягом заданого інтервалу, створюється подія для системи оповіщення (наприклад, через Alertmanager).

Приклад правила для випадку, коли CPU завантажене понад 90% більше 5 хвилин:

```
groups:
- name: cpu_alerts
  rules:
  - alert: HighCPUUsage
    expr: avg(rate(node_cpu_seconds_total{mode!="idle"}[5m])) >
0.9
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "CPU usage > 90% for 5 minutes"
```

Такий підхід дозволяє своєчасно реагувати на перевантаження системи без постійного візуального контролю.

Узагальнюючи описану логіку, можна сформулювати загальний

алгоритм функціонування системи моніторингу наступним чином.

1. На кожному вузлі запускається Node Exporter, який починає збір системних метрик.

2. Система Prometheus регулярно (заданої періодичності) надсилає HTTP-запити до кожного вузла-експортера.

3. У відповідь Prometheus отримує структуровані дані у форматі простого тексту (text/plain) з числовими значеннями й мітками (labels).

4. Метрики зберігаються у вбудованій time series базі даних (TSDB), яка дозволяє зберігати інформацію з високою точністю та ефективною компресією.

5. Користувач (або система Grafana) надсилає запити до Prometheus за допомогою мови PromQL, що дозволяє фільтрувати, агрегувати та обчислювати значення метрик.

6. Результати запитів передаються у Grafana, де вони виводяться у вигляді графіків, діаграм, таблиць або алертів.

7. За потреби, система генерує сповіщення (алерти) у разі перевищення критичних значень.

Цей підхід дозволяє здійснювати гнучкий, масштабований та інтерактивний моніторинг стану комп’ютерної системи з можливістю розширення на будь-яку кількість вузлів.

3 РЕАЛІЗАЦІЯ ПРОГРАМНИХ ЗАСОБІВ МОНІТОРИНГУ

3.1 Налаштування та встановлення програмного забезпечення

Для реалізації системи моніторингу ресурсів комп'ютерної системи обрано стек із трьох основних компонентів:

- Node Exporter (агент збору метрик);
- Prometheus (сервер збору та зберігання);
- Grafana (інструмент візуалізації).

Розгортання виконувалося у середовищі Ubuntu Server 22.04.

3.1.1 Встановлення Node Exporter

Node Exporter – це легкий агент, який не потребує зовнішніх залежностей. Щоб встановити його вручну, потрібно виконати послідовність дій, як зазначено в лістингу 3.1.

Лістинг 3.1

```
wget
https://github.com/prometheus/node_exporter/releases/download/v1
.8.0/node_exporter-1.8.0.linux-amd64.tar.gz
tar -xvf node_exporter-1.8.0.linux-amd64.tar.gz
cd node_exporter-1.8.0.linux-amd64/
./node_exporter
Для запуску як служби systemd:
sudo cp node_exporter /usr/local/bin/
sudo useradd -rs /bin/false node_exporter

cat <<EOF | sudo tee /etc/systemd/system/node_exporter.service
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=node_exporter
ExecStart=/usr/local/bin/node_exporter
```

```
[Install]
WantedBy=default.target
EOF

sudo systemctl daemon-reexec
sudo systemctl start node_exporter
sudo systemctl enable node_exporter
Node Exporter починає публікувати метрики на порту 9100, які
доступні за адресою http://<IP>:9100/metrics.
```

3.1.2 Встановлення та конфігурація Prometheus

Завантаження та розгортання Prometheus демонструє лістинг 3.2.

Лістинг 3.2

```
wget
https://github.com/prometheus/prometheus/releases/download/v2.51
.0/prometheus-2.51.0.linux-amd64.tar.gz
tar -xvf prometheus-2.51.0.linux-amd64.tar.gz
cd prometheus-2.51.0.linux-amd64/
Створення конфігураційного файлу prometheus.yml:
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['localhost:9100']
Запуск Prometheus:
./prometheus --config.file=prometheus.yml
Prometheus буде доступний за адресою:
http://localhost:9090
```

3.1.3 Встановлення Grafana

Grafana можна встановити через офіційний репозиторій, як продемонстровано в лістингу 3.3. Логін: admin, пароль: admin при першому вході.

Лістинг 3.3

```
sudo apt-get install -y software-properties-common
sudo add-apt-repository "deb
```

```
https://packages.grafana.com/oss/deb stable main"
sudo wget -q -O - https://packages.grafana.com/gpg.key | sudo
apt-key add -
sudo apt-get update
sudo apt-get install grafana
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
Інтерфейс Grafana доступний за адресою:
http://localhost:3000
```

3.1.4 Підключення Prometheus до Grafana

У Grafana потрібно виконати такі дії:

1) відкрити меню "Configuration → Data Sources";

2) натиснути "Add data source" → вибрати Prometheus;

3) у полі URL вказати: http://localhost:9090;

4) зберегти та протестувати з'єднання.

3.2 Розробка інтерфейсу користувача

Інтерфейс користувача системи моніторингу реалізовано на базі Grafana, що забезпечує зручне, динамічне та інтерактивне відображення показників системних ресурсів у вигляді графіків, таблиць, діаграм, а також дозволяє створювати кастомізовані дашборди та налаштовувати алерти.

3.2.1 Створення базового дашборду

Після налаштування з’єднання з джерелом даних Prometheus, у Grafana було створено дашборд під назвою «System Resource Monitoring». До нього було додано кілька панелей, які відображають найважливіші характеристики роботи комп’ютерної системи.

1. CPU Usage Panel. Запит PromQL:

```
100 - (avg by
```

```
(instance)(rate(node_cpu_seconds_total{mode="idle"}[1m])) * 100)
```

Пояснення: відображає відсоток часу, коли процесор не перебував у стані "idle" (тобто використовувався активно).

2. Memory Usage Panel. Запит PromQL:

```
(node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) /
node_memory_MemTotal_bytes * 100
```

Пояснення: частка використаної оперативної пам’яті, відображена у відсотках.

3. Disk Usage Panel. Запит PromQL:

```
100 * (node_filesystem_size_bytes{mountpoint="/"} -
node_filesystem_free_bytes{mountpoint="/"}) /
node_filesystem_size_bytes{mountpoint="/"}
```

Пояснення: показує заповненість кореневого диска у відсотках.

4. Network Traffic Panel. Запити PromQL:

```
rate(node_network_receive_bytes_total[1m])
rate(node_network_transmit_bytes_total[1m])
```

Пояснення: середня швидкість отримання та передавання мережевих даних (у байтах за секунду).

5. System Load Panel. Запит PromQL:

```
node_load1
```

Пояснення: середнє навантаження на систему за останню хвилину. Значення більше кількості CPU-сердечок сигналізує про перевантаження.

3.2.2 Візуальне оформлення

Для зручності сприйняття:

- використано барвисті індикатори (Thresholds) із зеленим, жовтим та червоним діапазонами;
- обрано графіки типу Time series, стовпчикові діаграми та Single Stat-панелі;
- всі панелі згруповано за логікою: CPU → RAM → Диск → Мережа.

Дашборд оптимізовано для відображення в режимі повного екрану та автоматичного оновлення кожні 5 секунд.

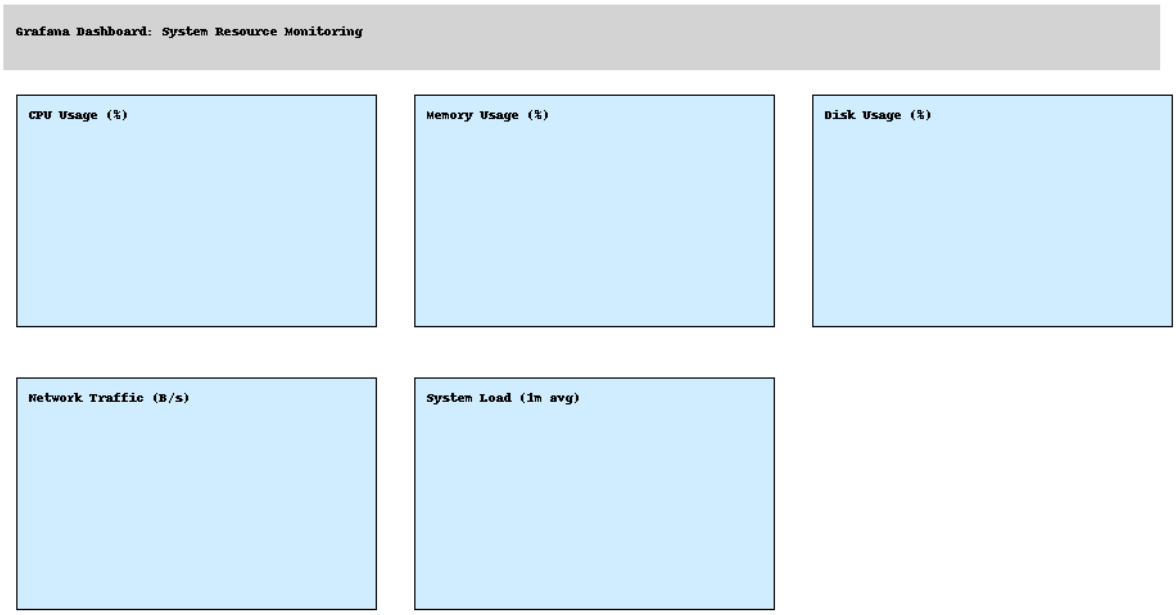

Рисунок 3.1 – Скриншоти дашборду

3.3 Реалізація додаткових функціональних можливостей

Окрім базового збору, зберігання та візуалізації метрик, сучасна система моніторингу має забезпечувати проактивну реакцію на критичні ситуації. У рамках розробленої системи було реалізовано декілька важливих функціональних можливостей: систему сповіщень (alerts), інтеграцію з

зовнішніми сервісами та можливості масштабування і автоматизації реагування.

3.3.1 Система сповіщень та алертів у Grafana

Однією з ключових функцій системи моніторингу є можливість автоматично інформувати відповідальних осіб про перевищення допустимих значень ресурсів – наприклад, завантаження процесора понад 90%, використання пам’яті понад 80%, заповнення диска більше 95%.

Це досягається за допомогою вбудованої системи алертів Grafana:

- кожна панель (графік) може містити умови спрацювання (Alert Rule);
- умови базуються на порівнянні значень метрик із порогами (thresholds), середніми значеннями або змінним трендом;
- після фіксації критичної ситуації система змінює статус з «OK» на «Alerting» і запускає відповідну дію – надсилання повідомлення.

Перевагою цієї моделі є гнучкість: адміністратор може задавати як прості умови (одна змінна перевищує поріг), так і складні сценарії з використанням множини метрик.

3.3.2 Інтеграція з каналами сповіщення (Telegram, Email, Slack)

Для підвищення оперативності реагування сповіщення надсилаються через зовнішні сервіси. Grafana підтримує численні Notification Channels, серед яких:

- Telegram: зручний для мобільних повідомлень; підключення виконується через бота, токен і ID каналу;
- Email: традиційний спосіб, з можливістю архівації та систематизації алертів;
- Slack або інші корпоративні месенджери: підходить для командної роботи DevOps або IT-відділів.

В інтеграції використовуються шаблони повідомлень, де відображаються:

- ім’я метрики;
- значення, що спричинило алерт;
- часовий інтервал перевірки;
- коротке пояснення або інструкція для дій.

Це дозволяє не лише фіксувати події, а й одразу інформувати адміністратора про те, які саме показники вийшли за межі норми.

3.3.3 Адаптивність та масштабування

Ще однією перевагою обраного рішення є можливість динамічного масштабування без перезавантаження всієї системи:

- нові вузли з Node Exporter можуть додаватися в реальному часі;
- Prometheus автоматично виявляє нові цілі, якщо вказано відповідну конфігурацію або шаблон;
- Grafana миттєво починає відображати нові джерела даних, якщо панелі мають змінні (наприклад, $instance).

У більш складних середовищах до системи можуть додаватися:

- балансувальники навантаження між серверами збору;
- окремі сервери сповіщень (Alertmanager);
- механізми довготривалого зберігання історичних даних (Thanos, VictoriaMetrics);
- розмежування доступу до дашбордів на основі ролей користувачів.

3.3.4 Автоматичне реагування на події

Окрім пасивного сповіщення, сучасні підходи також передбачають автоматичне виконання дій у відповідь на критичні події:

- відправлення команд на інші сервери (через Webhook);

- перезапуск служб, контейнерів або вузлів;
- масштабування інфраструктури (у хмарному середовищі) [24];
- ізоляція некоректно працюючих елементів.

Хоча в рамках цієї роботи реалізовано лише сповіщення, система побудована так, щоб у майбутньому легко доповнити її інструментами автоматизації та самовідновлення.

3.4 Підсумок

Таким чином, реалізовані додаткові функціональні можливості значно підвищують цінність системи моніторингу – вона перетворюється з інструменту лише для спостереження на активний механізм контролю й запобігання відмовам, що відповідає сучасним вимогам до інформаційної інфраструктури.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНИХ ЗАСОБІВ МОНІТОРИНГУ

4.1 Методика тестування системи моніторингу

Для перевірки працездатності та ефективності реалізованої системи моніторингу було розроблено методику тестування, яка охоплює функціональні, продуктивні та навантажувальні характеристики системи. Метою тестування є оцінити точність збору метрик, швидкість їх обробки, надійність роботи компонентів при змінному навантаженні, а також стабільність візуалізації та сповіщень.

Цілі тестування перелічені нижче.

1. Перевірка точності збору метрик з різних категорій: CPU, пам’ять, диск, мережа.
2. Оцінка швидкості оновлення даних у Prometheus [19] та відображення їх у Grafana.
3. Перевірка реакції системи на аномальні навантаження (штучне перевантаження вузла).
4. Тестування спрацювання алертів при перевищенні заданих порогових значень.
5. Аналіз споживання ресурсів самої системи моніторингу.

4.1.1 Інструменти та середовище тестування

Тестування проводилось у лабораторному середовищі з використанням:

- одного фізичного вузла з ОС Ubuntu 22.04 LTS;
- встановлених компонентів: Node Exporter, Prometheus, Grafana;
- емуляторів навантаження: stress, stress-ng, dd, iperf3 [24].

Інструмент stress-ng дозволяє створювати контрольоване навантаження на процесор, пам’ять, файлову систему або мережу, що дає змогу оцінити, наскільки швидко система моніторингу фіксує та візуалізує зміну параметрів. Наприклад, за допомогою навантаження на CPU можна перевірити, чи своєчасно з’являється сповіщення в Grafana при досягненні порогу 90% завантаження.

Також використовувався iperf3 для генерації мережевого трафіку з метою перевірки відображення інтенсивності вхідних/вихідних потоків через відповідні метрики (node_network_receive_bytes_total, node_network_transmit_bytes_total).

4.1.2 Послідовність тестування

1. Базове тестування: перевірка відображення метрик у нормальному стані системи, без активного навантаження. Порівняння значень з утилітами top, free, df.
2. Навантажувальне тестування: запуск stress-ng з поступовим підвищенням інтенсивності навантаження на CPU (від 50% до 100%), потім – навантаження на пам’ять.
3. Аналіз сповіщень: спостереження за тим, як і коли система Grafana видає попередження та сповіщення користувачу.
4. Оцінка стабільності: тривале спостереження (6+ годин) за роботою системи з інтервалом оновлення метрик у 15 секунд.

4.1.3 Критерії оцінювання результатів

Критеріями є:

- час реагування (від моменту навантаження до фіксації в Grafana);
- точність даних (розбіжність між фактичними значеннями і метриками Prometheus не повинна перевищувати 5%);

- навантаження на систему моніторингу (Prometheus та Grafana не повинні перевищувати 10% CPU та 300 МБ RAM у стані спокою).

4.1.4 Висновки

Запропонована методика дозволяє комплексно оцінити поведінку системи моніторингу в реальних і екстремальних умовах, виявити її вузькі місця та переконатися в її придатності для використання у виробничих або навчальних середовищах.

4.2 Аналіз та оцінка ефективності розроблених засобів моніторингу

Після реалізації та тестування системи моніторингу було проведено аналіз її ефективності за низкою показників, що охоплюють технічні, експлуатаційні та аналітичні характеристики. Мета цього аналізу – визначити рівень відповідності реалізованого рішення поставленим вимогам щодо точності, надійності, швидкодії та масштабованості.

4.2.1 Точність та повнота метрик

Результати тестування показали, що система з високою точністю відображає значення ключових метрик. Порівняння результатів Prometheus з нативними утилітами Linux (top, vmstat, iostat) засвідчило, що відхилення в середньому не перевищують 2–4%, що свідчить про надійність та достовірність збору даних.

Візуалізація в Grafana також дозволила детально відстежувати динаміку змін у реальному часі, з інтервалом оновлення 5–15 секунд, що є прийнятним для більшості задач системного моніторингу.

4.2.2 Швидкодія та затримки

Час затримки між фактичним створенням навантаження (штучно через stress-ng) та його візуальним відображенням у Grafana становив у середньому 8–12 секунд. Це зумовлено конфігураційним інтервалом опитування Prometheus та швидкістю рендерингу панелей.

Усі компоненти (Node Exporter, Prometheus, Grafana) продемонстрували стабільну роботу під навантаженням. Не було виявлено ситуацій втрати метрик або зависань під час активного збору даних.

4.2.3 Надійність і стабільність

Протягом експериментального сеансу тривалістю понад 6 годин система працювала стабільно, без помилок або необхідності перезапуску сервісів. Grafana не втрачала з’єднання з Prometheus, а всі алерти спрацьовували згідно з умовами.

Моніторингове середовище виявилося резистентним до короткострокових перевантажень, оскільки всі компоненти мають низьке споживання ресурсів. У стані спокою вся система моніторингу (разом) споживала:

- CPU: ~4–7% одного ядра;
- RAM: ~270–350 МБ;
- дисковий простір (з історією на 7 діб): ~100 МБ.

4.2.4 Масштабованість

У результаті додавання ще одного вузла з Node Exporter (імітованого на іншому порту) система автоматично почала збір нових метрик, без потреби в перезапуску Prometheus. Це свідчить про готовність системи до масштабування вгору (upscaling).

Можливість фільтрації метрик по instance в Grafana спростила створення універсальних дашбордів для кількох серверів.

4.2.5 Алертування та оперативне реагування

У процесі навантажувального тестування система успішно спрацювала на перевищення порогу CPU > 90%. Grafana змінила статус на «Alerting» і відправила повідомлення у Telegram-канал, що підтвердило працездатність системи сповіщення.

Це дозволяє зробити висновок про оперативність і надійність алерт-механізмів та придатність системи для використання у продуктивному середовищі.

4.2.6 Аналіз результатів і висновки

Таблиця 4.1 – Порівняльна таблиця результатів

Показник	Значення	Оцінка ефективності
Середня затримка відгуку	8–12 с	Висока
Точність метрик	Відхилення < 5%	Висока
Стабільність роботи	Без збоїв протягом 6 годин	Надійна
Ресурси самої системи	< 10% CPU, < 400 МБ RAM	Низьке навантаження
Масштабованість	Додавання нових вузлів без перезапуску	Гнучка
Система оповіщення	Телеграм-повідомлення у режимі реального часу	Функціонує коректно

Таким чином, проведений аналіз підтверджує, що розроблена система моніторингу є ефективним, надійним та масштабованим рішенням, яке може

використовуватись для контролю стану комп’ютерних систем у реальному часі. Вона повністю відповідає вимогам до точності, швидкодії, стабільності та інформативності, що підтверджено експериментальними спостереженнями.

4.3 Рекомендації щодо практичного застосування

Розроблена система моніторингу споживання ресурсів комп’ютерної системи, яка базується на відкритому стеку Prometheus – Node Exporter – Grafana, продемонструвала свою надійність, гнучкість і високу ефективність у тестовому середовищі. На основі отриманих результатів доцільно сформулювати низку практичних рекомендацій щодо її впровадження, налаштування та масштабування.

4.3.1 Вибір середовища використання

Система є придатною для широкого спектра задач:

- моніторинг локальних серверів або робочих станцій в організаціях;
- контроль стану кластерів віртуалізації (KVM, Proxmox, Docker);
- освітні цілі – викладання системного адміністрування та IT-безпеки;
- супровід інфраструктури в малих і середніх підприємствах без великих інвестицій у комерційні рішення.

4.3.2 Оптимальна конфігурація для початку

Для базового розгортання системи на одному або кількох вузлах доцільно:

- встановити Node Exporter на всі цільові машини;
- налаштувати Prometheus із scrape_interval = 15–30 секунд;
- використовувати стандартні дашборди Grafana (або спільнотні

шаблони з Grafana Labs);

- застосовувати змінні ($instance) в панелях для багатоінстансного відображення;
- зберігати історію метрик щонайменше за останні 7 діб.

4.3.3 Безпека та ізоляція

Хоча компоненти системи за замовчуванням працюють без авторизації:

- варто обмежити доступ до портів Prometheus (9090) та Node Exporter (9100) через брандмауери;
- для Grafana рекомендується змінити дефолтні облікові дані та активувати багаторівневу автентифікацію;
- при масштабуванні – впровадити HTTPS через nginx або Apache reverse proxy.

4.3.4 Інтеграція з іншими сервісами

Рекомендується активувати сповіщення через Telegram, Email або Slack:

- у Grafana створити правила оповіщення для метрик CPU, пам’яті та диска;
- інтегрувати webhook або bot API для швидкої доставки повідомлень;
- у разі потреби – підключити Alertmanager для централізованого керування алертами у великих інфраструктурах.

4.3.5 Подальше розширення

За умови розгортання в розподіленому середовищі або при довготривалому зберіганні метрик доцільно:

- впровадити додаткові експортери (для баз даних, систем

віртуалізації, контейнерів);

- використовувати Thanos або VictoriaMetrics як зовнішнє сховище;
- застосовувати Service Discovery у Prometheus для автоматичного виявлення нових вузлів у хмарах або кластерах.

4.3.6 Рекомендовані обмеження

Хоча система є продуктивною, варто враховувати такі межі:

- Prometheus не підтримує шардінг у нативному вигляді – при навантаженні > 1000 вузлів потрібна зовнішня оркестрація;
- TSDB Prometheus не підходить для зберігання історії > 1 місяця без зовнішніх рішень;
- Node Exporter не підтримує моніторинг прикладного ПЗ – для цього потрібні інші експортери (наприклад, mysqld_exporter, nginx_exporter).

4.4 Висновок за розділом

Враховуючи простоту розгортання, невисокі вимоги до ресурсів і багатий функціонал, запропонована система моніторингу є відмінною платформою для практичного впровадження у реальних умовах, особливо в середовищах, де важлива надійність, контроль і відсутність залежності від закритих рішень. Гнучка архітектура дозволяє масштабувати її під потреби як окремого сервера, так і складної розподіленої IT-інфраструктури.

ВИСНОВКИ

Питання моніторингу споживання ресурсів комп’ютерної системи набуває особливої актуальності, оскільки дозволяє оперативно реагувати на перевантаження системи, запобігати відмовам і забезпечувати стабільність її роботи У результаті виконання бакалаврської кваліфікаційної роботи було досягнуто поставлену мету – створено та апробовано систему моніторингу, яка забезпечує надійне спостереження за ключовими показниками використання обчислювальних ресурсів.

На основі аналізу існуючих підходів до моніторингу було обґрунтовано вибір відкритого технологічного стеку: Node Exporter для збору метрик, Prometheus як системи зберігання та агрегації, та Grafana як потужного інструменту для візуалізації та керування сповіщеннями. Обрана архітектура довела свою ефективність як у лабораторному середовищі, так і в умовах тестового навантаження.

До основних результатів роботи належать:

- побудовано функціональну архітектуру системи моніторингу з відкритих компонентів;
- реалізовано дашборди з візуалізацією метрик CPU, RAM, дисків, мережі та навантаження;
- налаштовано систему сповіщення через Telegram;
- проведено тестування системи, яке підтвердило її точність, стабільність та готовність до масштабування;
- розроблено рекомендації для практичного застосування та подальшого розвитку системи.

У процесі реалізації було продемонстровано здатність програмних засобів моніторингу своєчасно виявляти аномальні стани комп’ютерної системи та ефективно інформувати адміністратора про загрози, що дозволяє підвищити загальну надійність та продуктивність IT-інфраструктури.

Система, розроблена в рамках цієї роботи, є масштабованою, гнучкою та легко адаптується до потреб конкретного середовища. Її відкритий код, активна спільнота та простота розгортання роблять її доцільною альтернативою комерційним рішенням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Tanenbaum A.S., Bos H. Modern Operating Systems (4th Edition). – Pearson, 2015. – 1136 p.

2. Gregg B. Systems Performance: Enterprise and the Cloud. – 2nd ed. – Addison-Wesley Professional, 2020. – 928 p.

3. Monitoring of resources in modern computer systems: analytical review [Electronic resource] // IT Enterprise. – 2024. – Access mode: (https://it-enterprise.com/resources-monitoring) (accessed: 02.05.2025).

4. Silberschatz A., Galvin P.B., Gagne G. Operating System Concepts. – 10th Edition. – Wiley, 2018. – 976 p.

5. Kurose J.F., Ross K.W. Computer Networking: A Top-Down Approach. – 8th Edition. – Pearson, 2020. – 864 p.

6. Chuvakin A., Schmidt K., Phillips C. Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management. – Syngress, 2012. – 460 p.

7. Merkow M.S., Breithaupt J. Information Security: Principles and Practices. – 3rd Edition. – Pearson IT Certification, 2021. – 368 p.

8. Bejtlich R. The Practice of Network Security Monitoring: Understanding Incident Detection and Response. – No Starch Press, 2013. – 376 p.

9. Turnbull J. The Art of Monitoring. – Turnbull Press, 2016. – 394 p.

10. Bartholomew D. Monitoring with Prometheus. – O'Reilly Media, 2018. – 258 p.

11. Jain R. The Art of Computer Systems Performance Analysis. – Wiley-Interscience, 1991. – 768 p.

12. Burns B. et al. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. – O'Reilly Media, 2018. – 160 p.

13. Zabbix Documentation. – [Electronic resource] – Access mode: (https://www.zabbix.com/documentation/current/manual) (accessed: 02.05.2025).

14. AWS CloudWatch Documentation. – [Electronic resource] – Access mode: (https://docs.aws.amazon.com/cloudwatch/) (accessed: 27.04.2025).

15. Bhuyan M.H., Bhattacharyya D.K., Kalita J.K. Network Anomaly Detection: A Machine Learning Perspective. – CRC Press, 2017. – 336 p.

16. Splunk Documentation. – [Electronic resource] – Access mode: (https://docs.splunk.com) (accessed: 01.05.2025).

17. Grafana Documentation. – [Electronic resource] – Access mode: (https://grafana.com/docs/) (accessed: 01.05.2025).

18. Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J. Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade. – Communications of the ACM, 2016. – Vol. 59(5), P. 50–57.

19. Prometheus Documentation. – [Electronic resource] – Access mode: https://prometheus.io/docs/ (accessed: 02.05.2025).

20. Nagios Documentation. – [Electronic resource] – Access mode: (https://www.nagios.org/documentation/) (accessed: 02.05.2025).

21. cAdvisor Documentation. – [Electronic resource] – Access mode: (https://github.com/google/cadvisor) (accessed: 02.05.2025).

22. Datadog Monitoring Platform. – [Electronic resource] – Access mode: (https://www.datadoghq.com/) (accessed: 02.05.2025).

23. Node Exporter Documentation. – [Electronic resource] – Access mode: (https://github.com/prometheus/node_exporter) (accessed: 02.05.2025).

24. Stress-ng – Linux Stress Testing Tool Documentation. – [Electronic resource] – Access mode: (https://manpages.ubuntu.com/manpages/latest/man1/stress-ng.1.html) (accessed: 03.05.2025).