ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти магістр

на тему: «**Організація віртуалізованої системи розподілених обчислень**»

Виконав: здобувач вищої освіти
за освітньо-професійною програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
освітнього ступеня магістр
групи 126ІСТ_мд_21
Савченко О.А.
Керівник: Протас Н.М.
Рецензент: Біловод О.І.

Полтава – 2023 року

ВСТУП

Актуальність теми. Наразі, для корпоративних програм все частіше використовуються вебтехнології. Інфраструктура, необхідна для розробки, впровадження, хостингу та обслуговування програм, постійно ускладнюється і масштабується. Зі зростанням використання вебдодатків зростають і вимоги до серверних технологій. Бізнес все частіше очікує, що системи функціонуватимуть неперервно [1]. Витрати на забезпечення такої постійної доступності можуть досягати мільйонів доларів. Навіть короткий збій може призвести до серйозних фінансових втрат та ушкодити репутацію компанії.

При виборі стратегії забезпечення високої доступності головним чином потрібно враховувати профіль і характеристики конкретного бізнесу: оцінювати можливі втрати через прості в системі або втрати даних, прийнятний максимальний час простою [2].

Розподілені системи сприяють створенню інфраструктури, яка забезпечує безперервну роботу систем високої доступності. Одним з напрямків розподілених систем є кластерні технології. Використання кластера як інструменту для забезпечення високої надійності системи є дуже ефективним методом [3]. За допомогою спеціалізованого програмного забезпечення, відповідної інфраструктури, дублювання компонентів та балансування навантаження, кластер забезпечує безперервну роботу додатків. Тому питання побудови ефективних розподілених систем за кластерною архітектурою з використанням технологій віртуалізації залишається актуальним.

Зв'язок роботи з науковими програмами, темами. Робота відповідає дослідженням в межах науково-дослідної роботи «Розвиток підприємництва: управлінські, економічні, інноваційна та правові аспекти» відповідно до договору №9 від 15.05.2023 р. між ТОВ «ПАФ Гарант» та Полтавським державним аграрним університетом (розділ «Обгрунтування показників оцінювання гарантоздатності розподілених інформаційних систем»).

Метою кваліфікаційної роботи є забезпечення високої якості обслуговування

за рахунок розгортання програмного комплексу на основі віртуалізованої системи для вирішення різних обчислювальних задач з використанням принципів розподілених обчислень.

Завданнями кваліфікаційної роботи є:

– аналіз розвитку галузі та актуального стану розподілених та хмарних обчислень,

– визначення особливостей апаратних та програмних компонент систем розподілених обчислень,

– налаштування реалізації віртуалізованої розподіленої обчислювальної системи.

Об'єктом дослідження є процеси функціонування віртуалізованої розподіленої обчислювальної системи.

Предметом дослідження є вебкластер на основі віртуальних машин з підтримкою реплікації, кешування та балансування навантаження.

Методи дослідження – проведені в роботі дослідження базуються на методах системного аналізу, систем масового обслуговування, які використовувалися при розробці налаштувань вебкластеру як віртуалізованої розподіленої обчислювальної системи.

Інформаційна база кваліфікаційної роботи складається з наукових статей, міжнародних аналітичних видань і звітів, матеріалів наукових конференцій інтернет-ресурсів, що місять інформацію про архітектуру сучасних розподілених обчислювальних систем, а також даних, отриманих від провідних IT-компаній у сфері хмарних технологій.

Елементи наукової новизни полягають у розроблені архітектури вебкластеру на основі віртуальних машин у хмарі Amazon та досліджені працездатності системи при високому навантаженні та відмовах серверів бази даних.

Практична значущість роботи полягає в можливості повторного застосування та модифікації розроблених налаштувань вебкластеру при розгортанні високонавантажених вебсистем. Отримані результати можуть бути корисними для IT фахівців при впровадженні систем високої готовності з

використанням реплікації бази даних.

Апробація результатів дослідження відбувалася шляхом оприлюднення доповідей на наукових конференціях, семінарах.

Публікації. За результатами проведеного дослідження опубліковано тези: «Організація та планування завдань у віртуалізованих системах для розподілених обчислень», Матеріали XII Міжнар. наук. конференції «Інформаційні технології в енергетиці та агропромисловому комплексі», м. Львів, 04-06 жовтня 2023 р.; «Аналіз стану інформаційної безпеки: видів загроз і методів їх усунення», Матер. науково-практичної конференції за підсумками виробничої практики здобувачів вищої освіти спеціальності «Інформаційні системи та технології», 17 вересня 2023 р., м. Полтава.

Структура та обсяг кваліфікаційної роботи логічно пов'язані з задачами досліджень. Робота містить перелік умовних позначень, вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг текстової частини дипломної роботи складає 81 сторінка формату А4. Вона містить 36 рисунків і 1 таблиця. В роботі використано 46 науково-технічних джерела.

РОЗДІЛ 1
АНАЛІЗ РОЗВИТКУ ГАЛУЗІ ТА АКТУАЛЬНОГО СТАНУ РОЗПОДІЛЕНИХ ТА ХМАРНИХ ОБЧИСЛЕНЬ

1.1 Розвиток обчислювальної техніки та розподілених обчислювальних систем

Протягом останніх років швидкість розвитку комп'ютерних технологій досягла неймовірних висот: від машин, що коштували 100 мільйонів доларів і здійснювали лише одну операцію в секунду, до машин, вартістю тисячі доларів, які виконують 10 мільйонів операцій в секунду. Різниця у співвідношенні вартості та продуктивності становить 10^{12}.

Однією з ключових причин такого прискореного росту обчислювальної техніки став винахід високошвидкісних комп'ютерних мереж. Локальні мережі LAN (Local-Area Networks), які об'єднують сотні комп'ютерів, дозволяють машинам обмінюватися короткими інформаційними повідомленнями за кілька секунд [2]. Швидкість передачі великих обсягів даних між станціями у локальних та глобальних мережах (WAN, Wide-Area Networks) становить від десятків до тисяч Мбіт/с [3].

Сьогодні завдяки цим технологіям досить просто створити комп'ютерну систему з великою кількістю станцій, які будуть з'єднані високошвидкісною мережею. Ця система може називатися комп'ютерною мережею або розподіленою системою (distributed system) [4].

Сучасні розподілені обчислювальні системи демонструють швидкі зміни в організації ідеологій та підходів. Протягом короткого періоду, що вони існують, було створено велику кількість моделей, реалізуючи різні підходи до розподілених обчислень. Нові та трендові підходи постійно замінюють популярні схеми минулого, часто перетинаючись з базовими ідеями в розробці.

До середини 90-х років були розроблені дві концепції розподілених

обчислювальних систем: Вебпідхід, орієнтований на інформаційний простір для користувачів; технології розподілених об'єктів (CORBA та DCOM), спрямовані на створення розподілених середовищ, що дають доступ до мережевих ресурсів та імітують локальні додатки [5].

За останнє десятиліття спостерігається активний розвиток нових методів для розподілених обчислювальних систем та програмного забезпечення на проміжному рівні. Розвиток технологій однорангових мереж (peer-to-peer, P2P) та грід-технологій став ключовим. Використання однорангових мереж розширило можливості багатьох користувачів, дозволяючи надавати, а не лише споживати інформацію. Грід-технології сприяли впровадженню великих комплексів, які обробляють та зберігають дані, спочатку опираючись на принцип «обчислення за вимогою» [6, 7].

За допомогою сервіс-орієнтованої архітектури та вебсервісів вдається виправляти недоліки попередніх розподілених об'єктних технологій. Об'єднання бізнес-підходів до організації обчислювальних ресурсів у формі сервісів разом з ідеями грід-обчислень вплинуло на зародження нової концепції «Хмарних обчислень» [8].

1.2 Поняття розподілених обчислень та розподіленої системи

Розподілені обчислювальні системи (англ. Distributed Computing), є складовою частиною теорії обчислювальних систем. Вони вивчають теоретичні аспекти організації розподілених обчислень.

Визначення розподілених систем має різноманітні формулювання, але жодне з них не є строгим чи загальноприйнятим. Леслі Лемпорт, американський вчений у галузі обчислювальних систем, описав розподілену систему як таку, що користувач розуміє як розподілену, коли відмова одного комп'ютера, навіть не пов'язаного з його роботою, призводить до зупинки всієї системи [9]. Це визначення підходить для багатьох розподілених систем, особливо для тих, де єдине місце відмови може

призвести до зупинки системи (Single Point of Failure).

Ендрю С. Таненбаум, американський професор обчислювальної техніки, описав розподілену систему як набір незалежних комп'ютерів, які для користувача виглядають як єдина об'єднана система [10]. Оскільки самі комп'ютери не можуть утворювати цілісну систему, необхідним є використання проміжного програмного забезпечення (Middleware). Це дозволяє приховати відмінності між комп'ютерами та їхніми технологіями зв'язку для користувача, створюючи враження однієї системи.

Також варто взяти до уваги визначення, запропоновані іншими дослідниками:

Розподілена система – набір незалежних комп'ютерів, які не мають спільної пам'яті чи загального часу [11]. Взаємодія відбувається через передачу повідомлень по комунікаційній мережі, де кожен комп'ютер має свою власну пам'ять і операційну систему, але спільно працює над загальним завданням.

Розподілена система – система, в якій взаємодія та синхронізація програмних модулів відбуваються через передачу повідомлень між незалежними комп'ютерами мережі [12].

Поняття «розподілена система» охоплює широкий спектр від слабо зв'язаних багатопроцесорних комплексів до жорсткопов'язаних багатомашинних систем [13]. У апаратній частині розподілені системи представлені незалежними комп'ютерами чи процесорами, а програмна частина – автономними процесами. Всі ці компоненти (процеси, комп'ютери, процесори) є вузлами розподіленої системи. Для автономності, процесорам потрібне власне незалежне управління, тому паралельні комп'ютери, засновані на принципі «одна команда для багатьох даних» (SIMD – Single Instruction – Multiple Data) [14, 15], не вважаються розподіленими системами. За умови, що будь-який процес має свій власний стан і цей стан ізольований від інших, він вважається автономним процесом. Різні вузли розподіленої системи працюють з різною швидкістю, і час доставки повідомлень є непередбачуваним.

Оскільки процеси є вузлами системи, програмні системи, які об'єднують

взаємозалежні процеси, що працюють на одному обчислювальному пристрої, також можуть бути розглянуті як розподілені системи. У цьому випадку передача повідомлень відбувається через канали взаємодії з розділеною пам'яттю, а не за допомогою мережі зв'язку. Однак для більшості розподілених систем характерним залишається наявність кількох процесорів, що взаємодіють через засоби комунікації. На рис. 1.1 зображено типову розподілену систему [16].

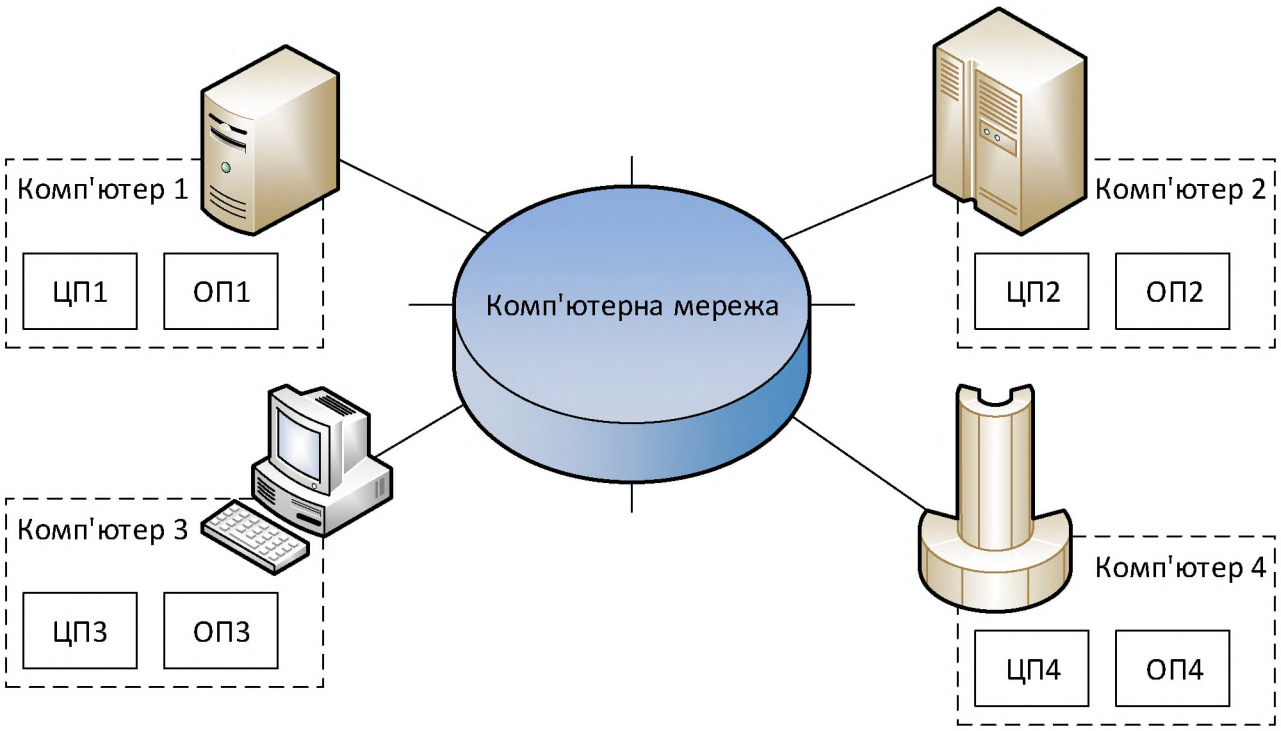


Рисунок 1.1 – Об'єднання незалежних комп'ютерів засобами комп'ютерної мережі у розподілену систему

До характеристик, що відзначають розподілені системи, належать [17]:

– Відсутність синхронізації для вузлів розподіленої системи. Це пов'язано з територіальним розподілом процесорів системи, що призводить до асинхронності їх роботи.

– Відсутність спільної пам'яті. Це означає, що потрібен обмін повідомленнями серед вузлів для організації взаємодії та синхронізації між ними. Це також означає, що немає єдиного фізичного синхронізму для всіх процесорів. Деякі розподілені системи застосовують розподілену пам'ять DSM (Distributed

Shared Memory), щоб створити видимість єдиного адресного простору для всіх процесорів, де розподілена пам'ять може бути представлена як віртуальна пам'ять з оперативною пам'яттю віддаленої станції, а не власним диском.

– Географічний розподіл. Необов'язково об'єднання у глобальну обчислювальну мережу для розподілених систем. На сьогоднішній день кластер, що складається зі звичайних робочих станцій (COW – Cluster Of Workstation), які з'єднані локальною обчислювальною мережею (ЛВС), розглядається як розподілена система. Кластери COW стають більш популярними через низьку вартість компонентів та високу продуктивність. Прикладом такої технології є ядро пошукової системи Google.

– Незалежність та гетерогенність. У розподілених системах комп'ютери слабко пов'язані між собою, вони можуть відрізнятися за складом, продуктивністю, часом виконання завдань, операційними системами. Часто це різні частини різних систем, але виконують загальне завдання, працюючи взаємопов'язано та надаючи свої послуги одне одному.

1.3 Мета розгортання та задачі, що виконують сучасні розподілені системи

Останнім часом значущість та популярність розподілених систем лише зросли. Основними факторами, що вплинули на цей ріст, є [19]:

1. Географічно розподілене обчислювальне середовище. Більшість обчислювальних середовищ представлені територіально розподіленими системами. Наприклад, банківська мережа: міжбанківські транзакції здійснюються через взаємодію мереж банків між собою. Мережа Інтернет також є географічно розподіленим обчислювальним середовищем.

2. Потреба у збільшенні продуктивності обчислень. Швидкодія однопроцесорних систем досягає максимуму своїх можливостей. Суперскалярна архітектура, матричні та векторні процесори, однокристальні багатопроцесорні

системи підвищують продуктивність обчислювальних систем шляхом паралельного виконання команд. Ці методи дозволяють збільшити продуктивність вдесятеро у порівнянні з класичними послідовними рішеннями. Об'єднання численних процесорів у великі багатопроцесорні системи та багатомашинні комплекси спрямовані на збільшення продуктивності систем у тисячі разів та забезпечення хорошої масштабованості.

3. Спільне використання ресурсів. Однією з важливих цілей, яку прагнуть досягти розподілені системи, є забезпечення користувачам або застосуванням доступу до віддалених ресурсів (апаратних компонентів, програмних абстракцій) та можливість їхнього спільного використання.

4. Відмовостійкість. Характерною особливістю розподілених систем, що відрізняє їх від окремих комп'ютерів (поломка одного з вузлів призводить до зупинки всієї системи), є стійкість до часткових відмов [20]. Вийшовши з ладу одного або кількох компонентів, система продовжує функціонувати, хоч із незначним зниженням продуктивності. Це можливо завдяки апаратній або програмній надмірності: до системи додаються додаткові обладнання або процеси, що забезпечують коректне функціонування системи при несправності деяких компонентів. У таких ситуаціях розподілена система намагається приховати випадки відмов або помилок у деяких процесах з інших. Наприклад, у системах з потрійним модульним резервуванням TMR (англ. Triple Modular Redundancy) застосовуються три ідентичні обчислювальні модулі, які виконують одні й ті ж обчислення, коректний результат яких визначається простим голосуванням.

Існують 4 важливі завдання, для вирішення яких будуються розподілені системи. Розподілені системи дозволяють з'єднувати користувачів із ресурсами, успішно приховують той факт, що ресурси розкидані по мережі, можуть бути відкритими та масштабованими.

Основне завдання, яке ставиться перед розподіленими системами – полегшити користувачам доступ до віддалених ресурсів та забезпечити їхнє спільне використання, регулюючи цей процес [21]. Ресурси можуть бути як віртуальними, так і фізичними: принтери та комп'ютери, пристрої зберігання

даних, файли та дані, вебсторінки та мережі. Спільне використання ресурсів має низку причин свого існування. Наприклад, економічність: дешевше використовувати один принтер на групу користувачів, ніж забезпечувати кожного своїм. За тим же принципом розумніше використовувати дорогі ресурси – високопродуктивні сховища даних або суперкомп'ютери. Також з'єднання користувачів та ресурсів полегшує кооперацію та обмін інформацією. Яскравим прикладом є успіх Інтернету, де обмін даними здійснюється простими протоколами.

Розподілена система називається прозорою (англ. Transparent), коли вона представлена користувачам або додаткам єдиною комп'ютерною системою [22]. Основне завдання таких систем – приховувати факт того, що фізично процеси та ресурси знаходяться на безлічі комп'ютерів.

Види прозорості [17, 23]:

- прозорість доступу: приховується відмінність у поданні даних та доступі користувача до ресурсів;
- прозорість реплікації: ховаються випадки реплікації ресурсу;
- прозорість розташування: приховується фізичне розташування потрібного ресурсу в системі;
- прозорість паралельного доступу: приховується факт спільного використання ресурсів кількома користувачами;
- прозорість перенесення: ховаються випадки переміщення ресурсів, не впливаючи на доступ до них;
- прозорість зміни розташування: приховується відміна переміщення ресурсів під час обробки;
- прозорість відмови: маскує випадки збоїв у системі;
- прозорість безпеки: приховується місце зберігання ресурсу на диску чи у оперативній пам'яті.

Ступінь прозорості залежить від завдань конкретної розподіленої системи та може змінюватись в межах продуктивності та прозорості системи. Повне

приховування розподілу процесів чи ресурсів неможливе через обмеження передачі сигналів та затримки при доступі до ресурсів, розташованих віддалено. При виборі ступеня прозорості знаходиться компроміс між продуктивністю системи та її прозорістю.

Відкрита система – це система, що реалізує відкриті специфікації для інтерфейсів, служб та форматів даних, які підтримуються [24]. Це необхідно для:

– інтеграції розробленого прикладного ПЗ у широкому діапазоні систем з мінімальними змінами – мобільність та переносимість додатків;

– взаємодії з іншими прикладними додатками на локальних та віддалених платформах – здатність до взаємодії;

– взаємодії з користувачами у стилі, що полегшує їх перехід від системи до системи – мобільність користувача.

Ця концепція прийнята комітетом IEEE POSIX 1003.0 [25]. Ключовим у цьому є використання поняття «відкрита специфікація». Відкрита специфікація – це загальнодоступна, підтримувана відкритим та голосним погоджувальним процесом специфікація, постійно адаптована до нових технологій та відповідає загальноприйнятим стандартам [26, 27]. Вона є незалежною від конкретних технічних та програмних засобів виробників, однак однаково доступна всім зацікавленим сторонам, які можуть брати участь у її розвитку; контролюється громадською думкою.

Для розподілених систем якість відкритості досягається, коли специфікація та опис основних інтерфейсів програмних вузлів системи доступні розробникам, описані та опубліковані. Це стосується внутрішніх компонентів системи. Інтерфейси описуються за допомогою мови визначення інтерфейсів IDL (Interface Definition Language). У них вказуються назви доступних функцій, типи переданих параметрів, значень, що повертаються та інше. Служби, що надають ці інтерфейси, задаються неформально мовою опису.

Завдячуючи такому опису будь-який процес, якому потрібна певна служба, через відповідний інтерфейс може звернутися до іншого процесу, що відповідає за реалізацію цієї служби. Це створює можливість створювати кілька окремих

реалізацій однієї й тієї ж служби, працюючих абсолютно однаково для зовнішніх процесів. Це реалізує властивість переносимості [28].

Важливою перевагою відкритих розподілених систем є можливість їх створення з різнорідного апаратного та програмного забезпечення. Додавання нових вузлів чи заміна існуючих не впливає на інші вузли. Фізично це проявляється можливістю легко додавати нові станції або замінювати вже існуючі більш потужними. На програмному рівні це дозволяє легко інтегрувати нові сервіси або вже існуючі реалізації. Розширюваність – важлива характеристика розподілених систем.

Масштабованість системи – це здатність змінювати характеристики системи в залежності від кількості користувачів та підключених ресурсів, а також від ступеня географічної розподіленості системи [29]. Важливими параметрами є працездатність, ефективність, вартість, трудовитрати виробництва, налагодження, підтримка, управління та зручність роботи з системою. Для різних систем масштабованість досягається різними способами: для одних – підходить лінійна залежність, а для інших – показник не повинен змінюватись за будь-яких масштабних змін системи.

При створенні великих систем (сотні машин, тисячі користувачів) висока масштабованість досягається шляхом децентралізації основних служб та алгоритмів, реалізованих у цій системі. Існують кілька варіантів цього підходу [29]:

- децентралізація обробки запитів – використання кількох машин для обробки;
- децентралізація даних – розподіл даних у кількох сховищах або створення копій одного сховища;
- децентралізація алгоритмів роботи – використання алгоритмів, які не потребують повної інформації про поточний стан системи, і можуть працювати в експлуатації різних вузлів системи;
- використання асинхронного зв'язку – передача повідомлень без очікування відповіді;
- застосування змішаних систем організації взаємодії:

− ієрархічна організація систем для ефективного пошуку інформації та ресурсів;

− реплікація – створення копій даних у різних вузлах системи для балансування навантаження;

− кешування – запам'ятовування та зберігання запитів ближче до клієнта для швидкого доступу;

− забезпечення автономності взаємодіючих машин у системі від інших машин за принципом точка-точка P2P (peer-to-peer).

Коли параметри, що вирішуються системою з однаковою періодичністю, можна досить швидко збільшувати зі зростанням кількості існуючих ресурсів, то вважається, що система добре масштабується за продуктивністю. Ідеально, трудомісткість внесення змін при зростанні системи не повинна зростати. Чим швидше зростає число функцій при збільшенні кількості компонентів, задіяних системою, тим вище функціональність масштабування. Адміністративна масштабованість системи має велике значення. Це залежність зручності роботи з системою від кількості адміністративно незалежних організацій, зайнятих у її експлуатації.

1.4 Класифікація розподілених систем

Створення розподілених систем завжди зумовлене конкретними потребами та завданнями. Різноманіття розподілених систем виникає через різні методи їх створення та розвитку. Часто розглядають розподілені системи за кількома типами.

На сьогодні виділяють дві основні класифікації. По-перше, системи розподіленого типу класифікують за розмірами та способами управління [25, 30]:

1. Кластер – група комп'ютерів, що працюють в локальній мережі та адмініструються вручну.

2. Розподілені системи корпоративного рівня – великі групи комп'ютерів, що

об'єднують сотні машин і працюють за встановленими правилами під час спільної експлуатації ресурсів. Ці системи не потребують значних адміністративних зусиль, оскільки масштаб проектів зазвичай не дуже великий.

3. Глобальна система (грід-система) – мережа комп'ютерів, що охоплює весь світ, може складатись з мільйонів пристроїв. Адміністративне програмне забезпечення цих систем інтегроване в проміжне програмне забезпечення.

По-друге, розподілені системи поділяють за функціональними характеристиками: типом ресурсів, видами завдань та оптимізацією [31, 32]:

1. Обчислювальні системи (Computational Grid) – основна потужність системи зосереджена на потужності процесора.

2. Інформаційні системи (Data Grid) – основний ресурс – обсяг пам'яті даних.

3. Мережеві системи (Network Grid або Delivery Grid) – системи, де вузли функціонують як передавачі (router), щоб підвищити ефективність та стійкість мережі передачі даних.

Між вказаними типами систем немає чітких меж. Деякі приклади розподілених систем [33]:

- WWW – велика розподілена система для доступу до пов'язаних документів;
- Lotus Notes – система управління документами;
- eDonkey, eMule – системи для завантаження файлів;
- системи на основі проміжного програмного забезпечення, такі як Globus, gLite;
- системи, побудовані на IBM WAS, наприклад, Workplace Collaborative Learning; Їх сервери мають можливість розподілу робочого навантаження та кешування для оптимізації продуктивності.

Таким чином, використовуючи подвійну класифікацію можна чіткіше окреслити межі використання та застосовності системи.

1.5 Класифікація кластерних обчислювальних систем

Кластери високої доступності, позначені як «HA» (High Availability), спрямовані на забезпечення безперервності роботи служб або сервісів у випадку відмови декількох серверів. Вони гарантують продовження роботи шляхом використання надлишкової кількості компонентів у кластері, які забезпечують стабільну роботу. Мінімальна кількість вузлів для підвищення доступності становить 2. Існує безліч програмних рішень для створення кластерів високої доступності. Відомі три основних принципи створення відмовостійких кластерів та систем [33]:

– Холодний резерв – активний/пасивний вузли. Якщо активний вузол відмовляє, пасивний, який очікує на цю відмову, тут же відновлює роботу.

– Гарячий резерв – активний/активний вузли. Запити обробляються всіма вузлами, і при відмові одного, навантаження розподіляється між працездатними компонентами.

– Модульна надмірність – кожен запит дублюється на всіх вузлах. Цей принцип використовується, коли надійність системи абсолютно критична.

Ці принципи можна комбінувати та використовувати у різних поєднаннях для конкретних технологій.

Кластери розподілення навантаження, позначені як «NLB» (Network Load Balancing), розподіляють запити через один або кілька вхідних вузлів, які перенаправляють їх на інші обчислювальні вузли для обробки [34]. Їх головна мета – забезпечити продуктивність, в них також часто використовуються методи для підвищення надійності.

Серверні ферми – це обчислювальні кластери, які використовуються для наукових досліджень з метою досягнення високої продуктивності процесора та низької скритності мережі [35]. Шляхом розбиття завдання на паралельно виконувані гілки та обміну даними через мережу, ці кластери зменшують час обчислень.

Типова архітектура обчислювальних кластерів, наприклад, кластер Beowulf,

складається із комп'ютерів, об'єднаних мережами, такими як Ethernet, InfiniBand або Myranet, та використовують ОС Linux. Високопродуктивні кластери, HPC (High-performance computing cluster), часто виносяться на окремий рівень.

Системи розподілених обчислень (grid) подібні до кластерної технології, але відрізняються зниженою доступністю кожного вузла та нестабільністю. Грід-системи розбивають завдання на незалежні один від одного процеси [36].

Кластери серверів – це група серверів, які логічно об'єднуються, обробляють однакові запити та використовуються як єдиний ресурс [37]. Програмно-організовані кластери мають спеціальний програмний модуль для зв'язку між серверами, синхронізації даних та розподілу навантаження. Це рішення дозволяє об'єднувати кілька програмних серверів в один фізичний сервер для оптимізації та забезпечення доступності лише при частих змінах конфігурації серверів, що вимагають їх перезавантаження.

1.6 Хмарні обчислення

Протягом останніх років хмарні обчислення стали популярними. Інформацію про цю технологію можна знайти всюди: в Інтернеті, ЗМІ та телебаченні, де описуються доступні можливості та функції. Вираз «хмара» вже давно закріпився у галузі мережевих технологій, і на мережевих діаграмах відображається як складна обчислювальна інфраструктура, приховуючи свою внутрішню організацію.

Термін «хмарні обчислення» з'явився порівняно недавно. За аналізом, проведеним Google, наприкінці 2007 року популярний термін «Grid-обчислення» почав витіснятися терміном «хмарні обчислення», який став вагомим [38]. У початку 2008 року компанія IBM створила проект Blue Cloud і спонсорувала ініціативу Joint Research Initiative for Cloud Computing в Європі [39].

Сьогодні хмарні обчислення стали невіддільною частиною повсякденного життя кожного користувача Інтернету. За експертними оцінками, ця технологія значно знижує вартість бізнес-додатків для кінцевих користувачів. Незважаючи на

всю привабливість хмарних обчислень, досі відсутнє чітке визначення та розуміння того, як вони взаємодіють із поняттям «Grid-обчислення».

Визначення хмарних обчислень нестандартизоване, існує багато різних визначень від різних корпорацій та вчених, у наукових колах продовжуються дебати з цього приводу. У комерційних виданнях [36, 37] найчастіше акцентується на тому, що хмарні обчислення повинні бути доступні кінцевому користувачеві, тоді як у науковому співтоваристві визначення базується на архітектурних особливостях.

Хмарні обчислення об'єднують великий пул легко використовуваних віртуалізованих ресурсів, які можуть динамічно перерозподілятися для оптимального використання. Масштабованість та віртуалізація визначають характеристики цієї технології. Віртуалізація відокремлює базові апаратні та програмні інфраструктури, створюючи узагальнений шар ресурсів [38]. Масштабованість дозволяє динамічно конфігурувати ресурси під зростаюче або спадне навантаження. Об'єднання ресурсів апаратної та програмної частини з додатками, що представляються користувачеві як сервіси, є основною особливістю хмарних платформ (рис. 1.2).

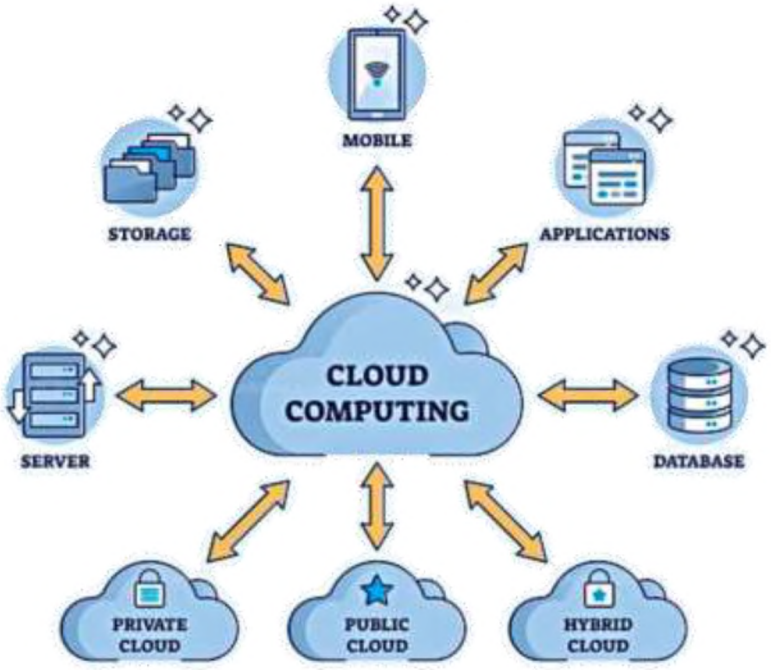


Рисунок 1.2 – Основні учасники хмарних обчислень

Основні характеристики хмарних обчислень включають в себе новий підхід до надання обчислювальних ресурсів, де інфраструктура та програми стають доступними у вигляді сервісів, що надаються споживачам зовнішніми постачальниками за принципом «плати за використання». Динамічна масштабованість та віртуалізація виступають основними атрибутами цієї технології, дозволяючи ефективно керувати ресурсами. Щодо методів доступу до хмарних послуг, кінцеві споживачі можуть користуватися спеціалізованими програмними інтерфейсами або веббраузерами.

На завершення можна сказати, що хмарні обчислення представляють собою перспективну технологію, що значно спрощує доступ до обчислювальних ресурсів, надаючи широкий спектр можливостей для бізнесу та користувачів у вигляді гнучкості та оптимізації використання обчислювальних потужностей.

Висновки до розділу 1

У першому розділі виконано аналіз сучасного стану галузі розподілених та хмарних обчислень. Визначення розподілених систем має різноманітні формулювання, але жодне з них не є строгим чи загальноприйнятим. Одне з визначень розподіленої системи – це система, в якій взаємодія та синхронізація програмних модулів відбуваються через передачу повідомлень між незалежними комп'ютерами мережі. Розподілені системи відрізняються від централізованих своєю асинхронністю в роботі вузлів, відсутністю спільної пам'яті та географічним розподілом, а також гетерогенністю та незалежністю компонентів, що можуть виконувати загальне завдання взаємопов'язано, надаючи послуги одне одному.

Проаналізовано основне завдання, яке ставиться перед розподіленими системами – полегшити користувачам доступ до віддалених ресурсів та забезпечити їхнє спільне використання, регулюючи цей процес. Розглянуто дві основні класифікації розподілених систем: за розмірами та способами управління, а також за функціональними характеристиками (типом ресурсів, видами завдань та

оптимізацією). Використовуючи подвійну класифікацію можна чіткіше окреслити межі використання та застосовності системи.

Системи розподілених обчислень можуть використовувати різні підходи, такі як кластери високої доступності з різними принципами відмовостійкості, мережеве розподілення навантаження та серверні ферми, щоб забезпечити стабільну та ефективну роботу у високонавантажених умовах. Архітектура кластерів, таких як Beowulf, та системи HPC використовуються для досягнення високої продуктивності, тоді як грід-системи розбивають завдання на незалежні процеси, забезпечуючи ефективне використання ресурсів.

Хмарні обчислення представляють собою перспективну технологію, що стала не тільки невід'ємною частиною повсякденного життя користувачів Інтернету, але й гнучким та потужним інструментом для бізнесу, який значно спрощує доступ до обчислювальних ресурсів та забезпечує ефективне їх використання, надаючи широкий спектр можливостей.

На основі проведеного аналізу сформульовано загальне завдання, яке полягає у розробці програмного комплексу на основі віртуалізованої системи для вирішення різних обчислювальних задач з використанням принципів розподілених обчислень. Загальне завдання розділено на три часткові задачі, дві з яких розглянуті у наступних розділах.

РОЗДІЛ 2
АПАРАТНІ ТА ПРОГРАМНІ КОМПОНЕНТИ СИСТЕМ РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ

2.1 Особливості апаратних компонент систем розподілених обчислень

Способи організації процесорів у розподілених системах різняться за типом з'єднання між процесорами та способами обміну повідомленнями. Незважаючи на багато варіантів організації комп'ютерних систем з декількома процесорами, жодна з них не досягла широкої популярності. Ці системи можна умовно розділити на дві групи: мультипроцесори, де пам'ять використовується спільно, та мультикомп'ютери, де кожна машина працює зі своєю пам'яттю (рис. 2.1). Мультипроцесори мають один адресний простір для всіх процесорів, що відрізняє їх від мультикомп'ютерів [39].

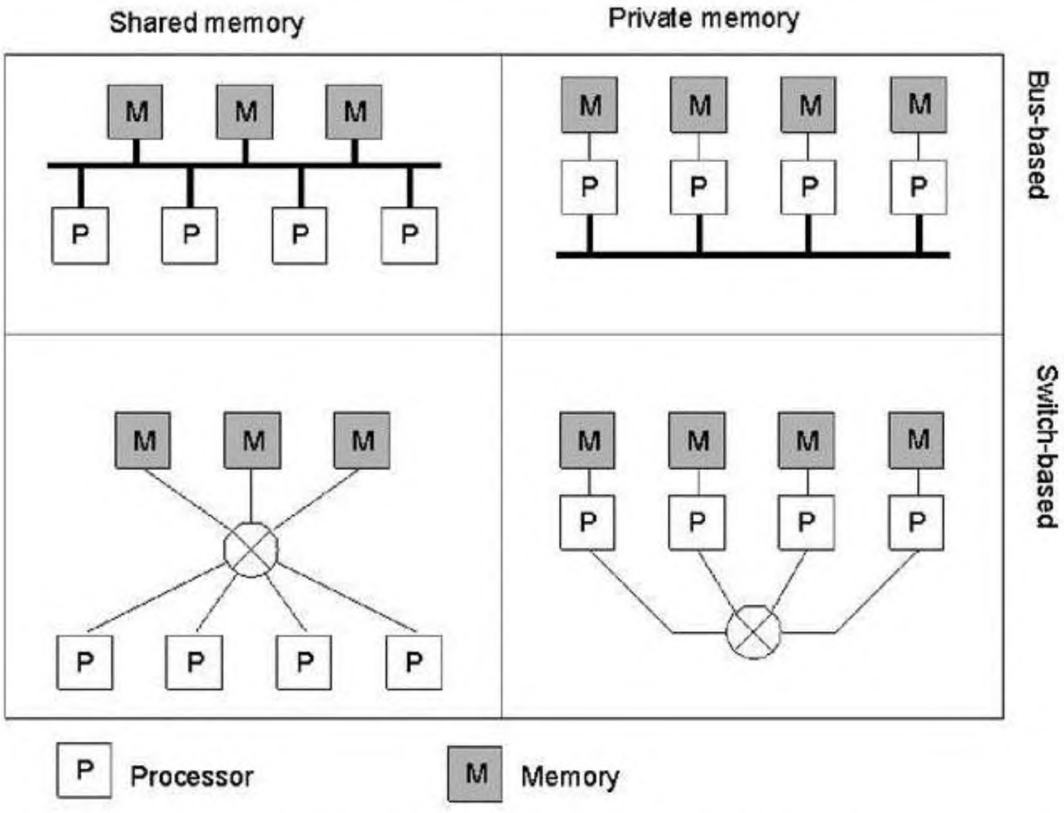


Рисунок 2.1 – Архітектури процесорів та пам'яті розподілених комп'ютерних систем

Ці дві групи можуть поділятися на підгрупи залежно від архітектури мережі, що об'єднує комп'ютери в єдину систему, такі як шинне об'єднувальне середовище. При класифікації багатопроцесорних систем важливим критерієм є організація оперативної пам'яті: симетричні мультипроцесорні системи, системи з масовим паралелізмом та NUMA-архітектура [40]. Сучасні високопродуктивні системи просуваються у чотирьох основних напрямках розвитку: векторно-конвеєрні суперкомп'ютери, симетричні мультипроцесорні системи, системи з масовим паралелізмом та кластери.

Однією з основних характеристик цих комп'ютерів є [41]:

– обробка потоку команд у конвеєрному режимі;

– існування системи управління набором векторних операцій, які працюють із цілими масивами даних.

Симетричні мультипроцесорні системи (SMP) будуються з однорідних процесорів та єдиного простору загальної пам'яті, до якої мають доступ всі процесори системи [42]. Хоча ця архітектура спрощує зв'язок між процесорами через загальну пам'ять, вона викликає ряд проблем, таких як конфлікти, що виникають при використанні загальної шини (рис. 2.2). Проте, її обмеження стосуються відмінностей у швидкості роботи процесорів та оперативної пам'яті, що вимагає використання кеш-пам'яті кожним процесором для згладжування цієї нерівності. Така практика збільшує витрати та обмежує продуктивність системи.

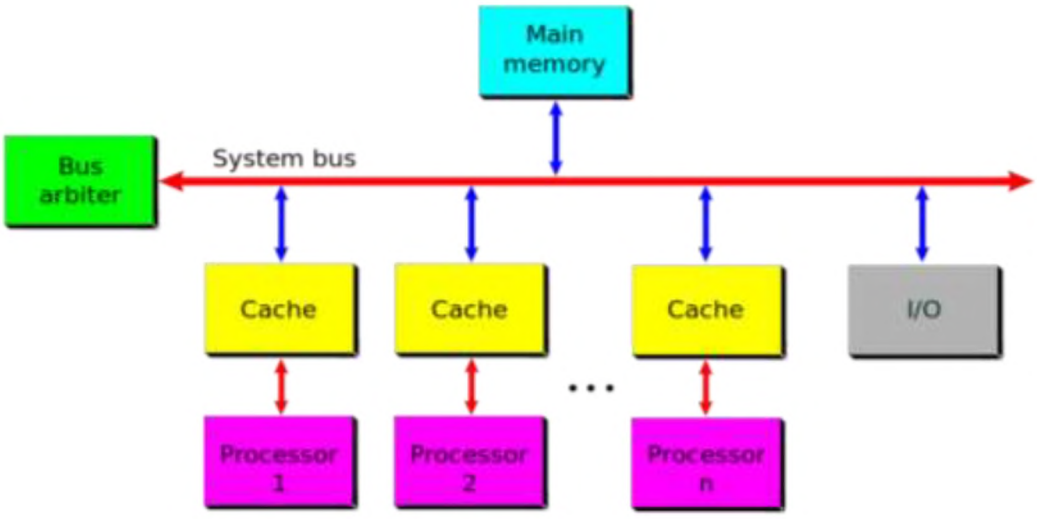


Рисунок 2.2 – Архітектура симетричної мультипроцесорної системи

Однією з труднощів є ієрархічна структура організації пам'яті: швидкість процесорів в рази перевищує швидкість оперативної пам'яті. Це ускладнює досягнення повної синхронізації роботи процесора і пам'яті до 100%. Для вирівнювання цих різниць у швидкості, кожен процесор має свою власну швидкісну кеш-пам'ять (буферну пам'ять), яка має продуктивність на рівні процесора. Використання таких мікропроцесорів у системах порушує основну концепцію SMP систем: рівність доступу кожного процесора до будь-якої області пам'яті. Організація апаратної підтримки взаємодії кеш-пам'яті дозволяє дотримуватися принципу загальної доступності, але при цьому виникає збільшення витрат та обмеження щодо підвищення продуктивності за рахунок нових компонентів. Типово симетричні мультипроцесорні системи обмежуються 32 процесорами. Для подальшого розширення вони використовують NUMA-технологію. Застосування цієї технології дозволяє використовувати до 256 процесорів. У випадку малих суперкомп'ютерів, цей тип систем ефективно працює як багатопроцесорні сервери з 2 або більше процесорами (може досягати 64).

Системи з масовим паралелізмом (MPP) представляють розподілені системи, де однакові обчислювальні вузли зв'язані спеціальним комунікаційним середовищем (рис. 2.3). Кожен вузол може функціонувати самостійно та виконувати повноцінну операційну систему, обмін даними відбувається за допомогою передачі повідомлень [43].

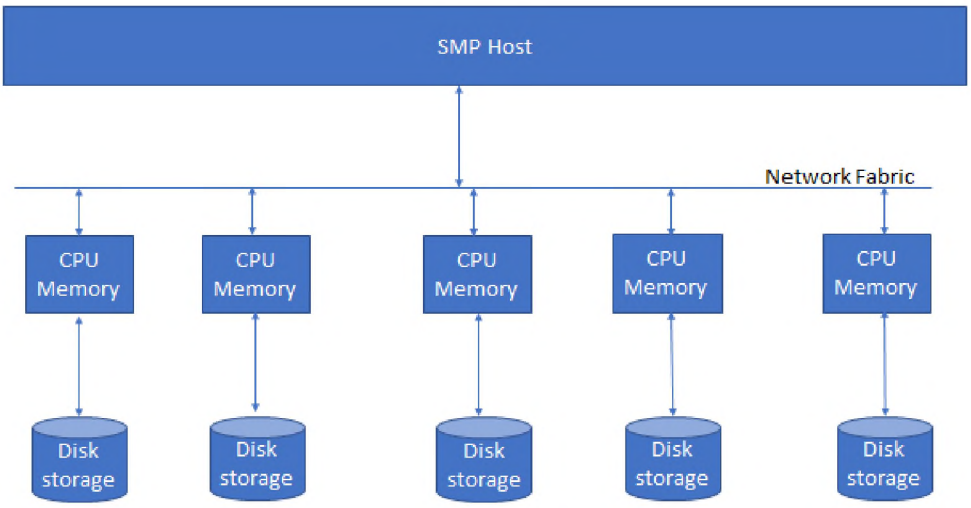


Рисунок 2.3 – Архітектура системи з масовим паралелізмом

Наявність в процесорі власної пам'яті та засобів для взаємозв'язку із вводом/виводом дозволяє вузлам цих систем працювати автономно. Кожен вузол може виконувати операційну систему або базові функції ядра. Звернення до пам'яті в цих системах відбувається за допомогою механізму передачі повідомлень. Цей тип систем має високий рівень масштабованості: досягнення необхідної продуктивності вимагає використання потрібної кількості вузлів (наприклад, Blue Mountain має 6144 вузли, ASCI Red – 9632 вузла) [44].

Важливий недолік систем MPP полягає у тому, що процесори обробляють дані на локальному рівні набагато швидше, ніж відбувається обмін цих даних між процесорами. Ця нерівність у швидкості призводить до складнощів у створенні програм, які забезпечать високу продуктивність, іноді це завдання стає надзвичайно складним або навіть неможливим.

Кластерні системи є наступним етапом розвитку архітектур масового паралелізму. Технологія кластерів дозволяє легко об'єднувати продуктивність кількох комп'ютерів для вирішення великих завдань шляхом розподілу завдань між вузлами та спільного використання ресурсів.

Оцінка й організація мультикомп'ютерних систем може бути складною. Гомогенні системи, такі як SAN (System Area Networks), мають просту структуру, де процесори мають прямий зв'язок зі своєю пам'яттю, але вимагають додаткового зв'язку між процесорами [45]. Вони використовують високошвидкісну мережу, що може бути шинною або комутаційною. Комунікація між процесорами може здійснюватися за допомогою мережі множинного доступу або шляхом маршрутизації у мережі, що може бути виконаною різними топологіями, такими як «решітка» або «гіперкуб» (рис. 2.4).

Решіткоподібна топологія добре підходить для двовимірних завдань, оскільки її простота сприяє створенню друкованих плат. Гіперкуб має форму куба з n розмірністю, де кожна вершина (процесор) з'єднана з іншою за ознакою відповідності. Для розширення гіперкуба в більш високі простори потрібно додати куби та з'єднати їх вершини.

У гомогенних системах MPP (масовий паралелізм) використовують

патентовані високошвидкісні мережі, що забезпечують малі затримки та високу пропускну здатність [46]. Кластери робочих станцій COW (Clusters Of Workstations) базуються на робочих станціях та персональних комп'ютерах, об'єднані за допомогою комерційних комунікаційних систем.

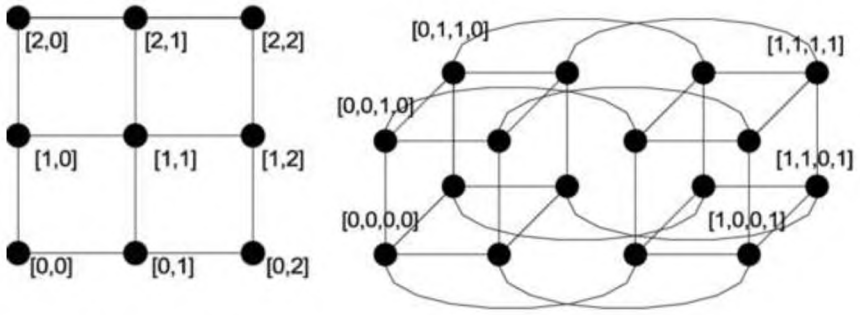


Рисунок 2.4 – Топології багатопроцесорних систем: решітка та гіперкуб

Гетерогенні системи складаються з різноманітних комп'ютерів, у яких різні типи процесорів, обсяги пам'яті та швидкості каналів введення/виведення. Ці системи вимагають глобального підходу до програмного шару, оскільки програми не можуть розраховувати на конкретну продуктивність чи доступ до певних сервісів у будь-який момент часу.

2.2 Архітектура програмних засобів розподілених обчислювальних систем

Для розподілених систем апаратна частина відіграє ключову роль, проте вигляд такої системи у реальності, передусім, залежить від програмного забезпечення. Принципи побудови розподілених систем дуже схожі на ті, що реалізовані в операційних системах: системи виконують роль менеджера ресурсів фізичного рівня, чиї компоненти сприяють спільному використанню різноманітних ресурсів (процесорів, пам'яті, периферії та інші) різними користувачами. Крім того, розподілені системи через віртуалізацію призначені приховувати від користувачів

складність організації апаратного рівня.

У випадку розподілених комп'ютерів операційні системи можуть бути слабко пов'язаними, у тому сенсі, що, працюючи разом, система виглядає як набір незалежних операційних систем, або сильно пов'язаними, коли робота відбувається з одним глобальним уявленням ресурсів.

Розподілені операційні системи DOS (Distributed Operating System) є прикладом сильно пов'язаних ОС і використовуються для управління гомогенними мультикомп'ютерними та мультипроцесорними системами. Основне завдання – приховати складнощі управління апаратними компонентами, до яких постійно звертаються процеси.

Мережеві операційні системи NOS (Network Operating Systems) є прикладом слабко пов'язаних систем і використовуються для гетерогенних мультикомп'ютерних систем. Основне завдання – організація доступу віддалених клієнтів до локальних служб.

Системи проміжного рівня (middleware) – це програмні інструменти, які спрямовані на кращу підтримку прозорості у розподілі.

Операційні системи у мультикомп'ютерних середовищах не потребують обов'язкового розміщення системних компонентів у загальній фізичній пам'яті, оскільки спільне використання цієї пам'яті не є критичним. Основний механізм взаємодії в цих системах – передача повідомлень. На рис. 2.5 представлена загальна структура таких операційних систем: система складається з вузлів, кожен з яких містить ядро з різними модулями для керування локальними ресурсами та модуль для обміну повідомленнями між вузлами (організація зв'язку). Програмне забезпечення загального призначення, що представляє операційну систему як віртуальну машину (дозволяє абстрагувати мультипроцесорну систему), знаходиться вище за рівнем локальних ядер та забезпечує паралельну обробку запитів. Це означає, що спільно використовувана пам'ять реалізується за

допомогою програмних модулів. На цьому рівні, як правило, використовуються кошти для розподілу завдань між процесорами, управління несправностями на апаратному рівні та забезпечення прозорості зберігання та обміну даними.

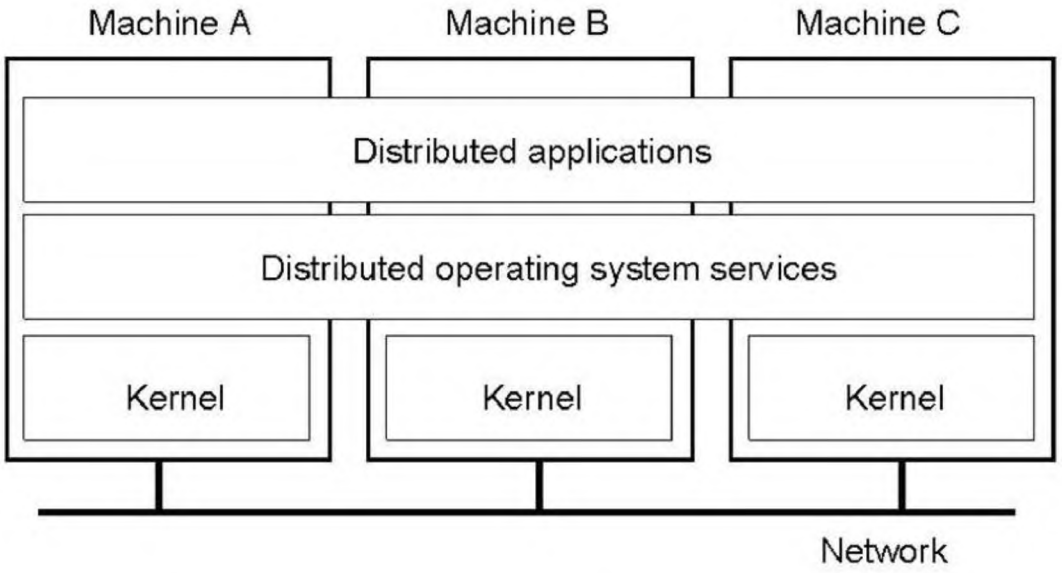


Рисунок 2.5 – Структура мультикомп'ютерних операційних систем

Для будь-якої розподіленої системи важливі різні критерії, які враховуються при створенні мультикомп'ютерних операційних систем. Відмінності між цими системами полягають у тому, що вони реалізовані на гомогенному і повністю керованому фізичному обладнанні.

Мережева операційна система (Network Operating System) – це програмне забезпечення, яке забезпечує обробку, зберігання та передачу даних в мережі інформації. Головні завдання мережевої ОС – розподіл ресурсів мережі, таких як дисковий простір, та управління мережею. Системний адміністратор налаштовує розділення ресурсів, встановлює паролі та права доступу для кожного користувача або групи користувачів. Це призводить до класифікації мережевих ОС на ті, які використовуються для серверів та для звичайних користувачів.

Мережева операційна система є основою будь-якої обчислювальної системи, яка складається з окремих комп'ютерів, з'єднаних у єдину мережу для обміну

повідомленнями та ресурсами за єдиною стандартною системою (протоколами). Ці протоколи відповідають за адресацію об'єктів, служби, захист даних та керування мережею (рис. 2.6).

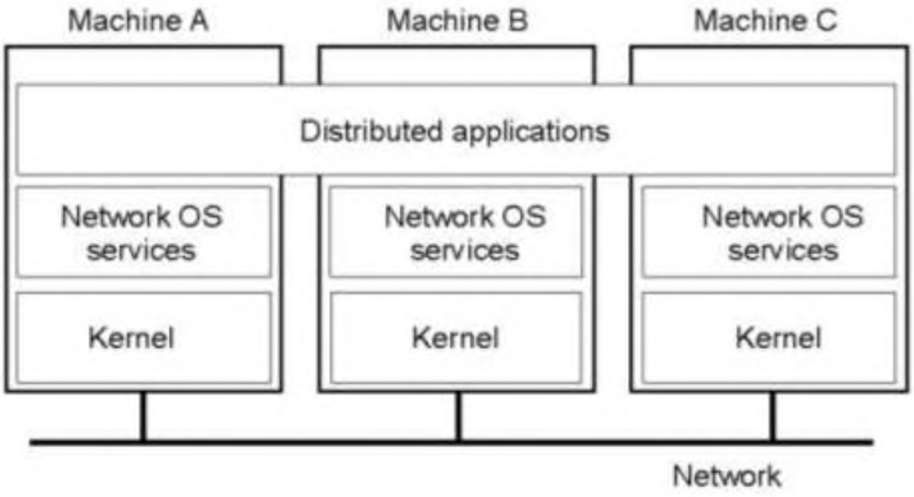


Рисунок 2.6 – Структура мережевих операційних систем

Існують два основних класи мережевих операційних систем: однорангові та дворангові, які відрізняються у розподілі функцій між комп'ютерами у мережі. Мережеві операційні системи, у порівнянні з розподіленими, мають більш просту структуру. Основна відмінність полягає в тому, що розподілені ОС використовують серйозні підходи для створення уявлення єдиної системи (повна прозорість).

Недоліками мережевих операційних систем є складність у використанні та управлінні, які потребують особливого підходу. Перевагою мережевих систем є можливість легко додавати, замінювати або видаляти нові комп'ютери завдяки їхній незалежності, достатньо підключити машину до загальної мережі та повідомити інші пристрої.

Обидва типи операційних систем, мережеві і розподілені, мають свої обмеження, які роблять їх неспроможними функціонувати як повноцінні розподілені системи. Перший тип систем не надає можливості створення єдиної

системи, а другий не призначений для управління групою автономних машин. Щоб компенсувати ці недоліки та підтримувати різноманітність комп'ютерів та різні методи зв'язку між ними, розподілені системи часто використовують спеціальне програмне забезпечення, яке поєднує переваги обох типів: з розподілених операційних систем – прозорість та порівняльна простота в експлуатації, з мережевих операційних систем – висока відкритість та ефективне масштабування. Це програмне забезпечення (рис. 2.7), що саме реалізує функціонал розподіленої системи, отримало назву програмного забезпечення проміжного рівня (англ. middleware).

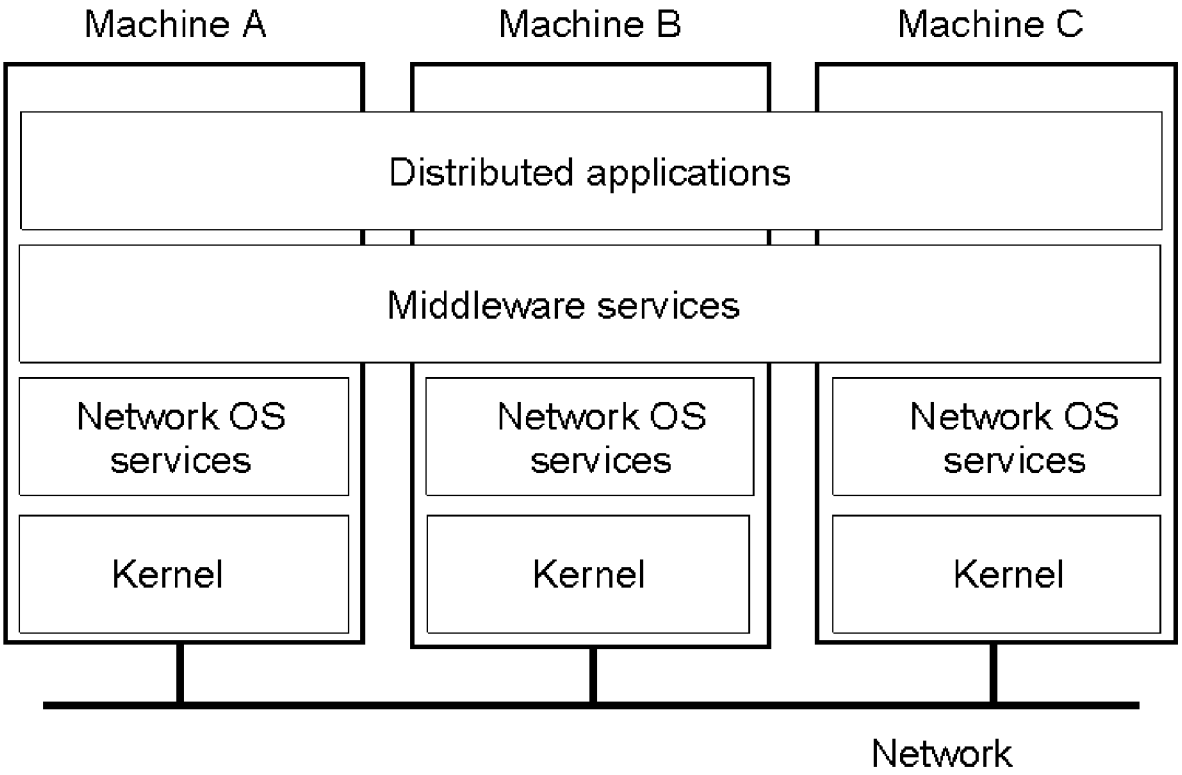


Рисунок 2.7 – Загальна структура систем з проміжним рівнем

Цей логічний програмний рівень призначений для додаткового абстрагування програм від базових платформ, приховання їхньої різноманітності від користувачів та додатків, а також забезпечення відповідної моделі програмування для розробників.

Зазвичай, програмне забезпечення проміжного рівня побудоване на певній концепції, що визначає розподіл та зв'язки всередині системи. Однією з простих схем є представлення об'єктів у вигляді файлів. Цей підхід дозволяє створювати програмне забезпечення проміжного рівня у вигляді розподіленої файлової системи. Це ПЗ виявилося добре масштабованим, але не суттєво випередило мережеві ОС: прозорість розподілу застосовувалася тільки до файлів, які призначалися для зберігання інформації.

Одним з важливих аспектів у цій моделі є використання віддалених викликів процедур (RPC – Remote Procedure Calls). Ця концепція маскує мережевий обмін: процес може викликати служби на віддалених станціях. Під час цього виклику параметри прозоро передаються до віддаленої станції для обробки запиту, після чого результати повертаються назад до викликаючої точки.

Концепція розподілених об'єктів полягає в тому, що ці об'єкти виконують інтерфейси, приховуючи внутрішні характеристики об'єкта. Це досягається шляхом розміщення об'єктів на одній зі станцій та надання доступу до їхніх служб іншим компонентам системи. При зверненні до методу об'єкта процес перетворюється на повідомлення, яке отримує об'єкт. Після обробки викликаного методу об'єкт повертає результати, які знову перетворюються на значення та передаються назад до процесу, який спочатку ініціював запит.

Склад стандартних служб:

1. Служба іменування: Це спільне використання та пошук вмісту, що є центральним для всіх систем проміжного рівня.

2. Засоби безпеки: Ці служби призначені для зберігання даних, що реалізується через розподілені файлові системи, інтегровані бази даних або спеціалізовані інструменти, що забезпечують взаємодію додатків та баз даних.

3. Служби для розподілених транзакцій: Вони застосовуються у випадках, коли збереження даних є важливим аспектом. Транзакції дають можливість для

неодноразових операцій читання/запису під час однієї сесії.

4. Засоби забезпечення захисту: Це типова служба більшості систем проміжного рівня, яка може гарантувати захист мережі за допомогою локальних операційних систем.

У середовищі проміжного рівня головним завданням програмного забезпечення є забезпечення прозорості доступу. Це досягається за допомогою використання високорівневих методів зв'язку, які приховують обмін повідомленнями на нижньому рівні. Методи підтримки зв'язку залежать від схеми розподілу, яку реалізує конкретне програмне забезпечення.

Клієнт-серверна архітектура – це модель взаємодії комп'ютерів та програм у мережі. Деякі машини у мережі мають ресурси (процесори, файлові системи, бази даних, поштові сервіси), тоді як інші можуть звертатися до цих ресурсів. Комп'ютери, що керують ресурсами, вважаються серверами, тоді як ті, що користуються цими ресурсами, – клієнтами. За такою класифікацією програми також можна поділити: серверна програма виконує операції та надає певний набір служб, тоді як клієнтська програма користується наданими службами.

Клієнти та сервери, разом із базовою операційною системою та середовищем взаємодії, утворюють єдину систему, що організовує розподілені обчислення та забезпечує аналіз та подання даних. У випадку стійкої мережі зв'язок між клієнтом та сервером може здійснюватися за допомогою простого протоколу, який не вимагає встановлення з'єднання. При зверненні до служби, клієнт перетворює запит на повідомлення, вказуючи необхідну службу та вхідні дані, та відправляє це повідомлення на сервер. Сервер, постійно очікуючи вхідних повідомлень, обробляє їх та відправляє результати назад у вигляді відповідного повідомлення (рис. 2.8).

Використання протоколу запит-відповідь, коли немає потреби у підтримці з'єднання, є ефективним для взаємодії. Однак можливі труднощі при пошкодженні або втраті повідомлень через випадкові збої зв'язку.

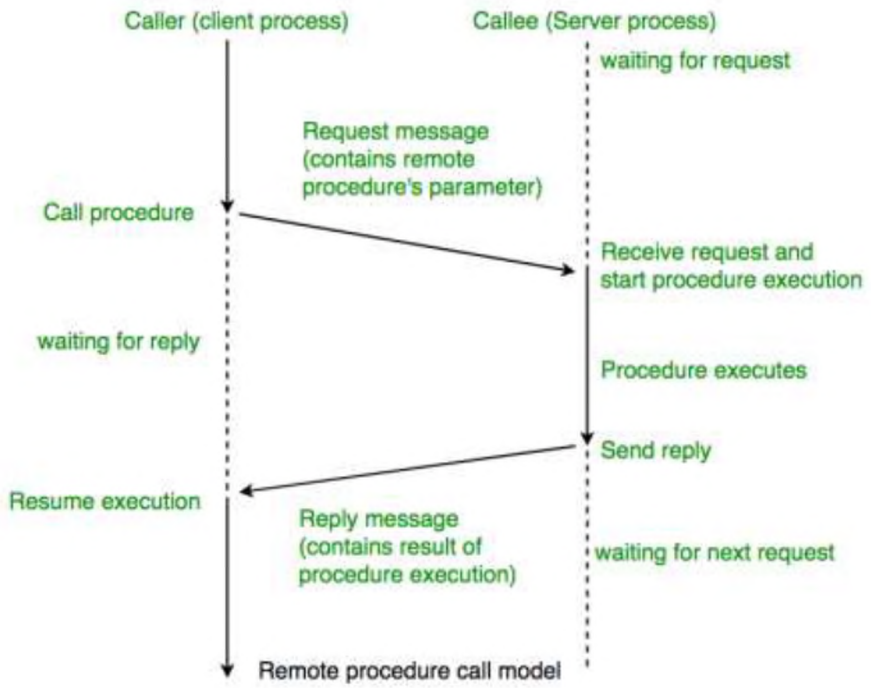


Рисунок 2.8 – Схема взаємодії клієнта та сервера

Для систем, які потребують високої стійкості, застосовують протоколи з встановленням з'єднання: клієнт спочатку встановлює з'єднання з сервером перед надсиланням запиту, а сервер після відповіді розриває з'єднання. Цей підхід має свої переваги та недоліки: він забезпечує стабільність, але може виявитися менш продуктивним та вимагати більше ресурсів для установки та розриву з'єднання.

2.3 Багаторівневі програмні архітектури розподілених обчислень

Програми складаються з трьох логічних компонентів, призначення яких відповідає класифікації функцій:

– компонент представлення: включає функції введення та відображення даних;

– прикладний компонент: охоплює прикладні функції;

– компонент доступу до інформаційних ресурсів: включає менеджер

ресурсів, а також функції зберігання та управління даними.

Існують три моделі підходів до реалізації додатків:

– RDA-модель – доступ до віддалених даних (англ. Remote Date Access);

– DBS-модель – сервер бази даних (англ. DateBase Server);

– AS-модель – сервер додатків (англ. Application Server).

RDA-модель (доступ до віддалених даних): у цій моделі комп'ютер-клієнт виконує прикладні та функціональні завдання введення та відображення даних. Доступ до ресурсів здійснюється через оператори спеціальної мови або викликами функцій спеціальної бібліотеки (рис. 2.9). Запити до ресурсів передаються через мережу до сервера, який відповідає за обробку та виконання запитів. Після виконання операцій з запитом, сервер повертає клієнту блоки даних.

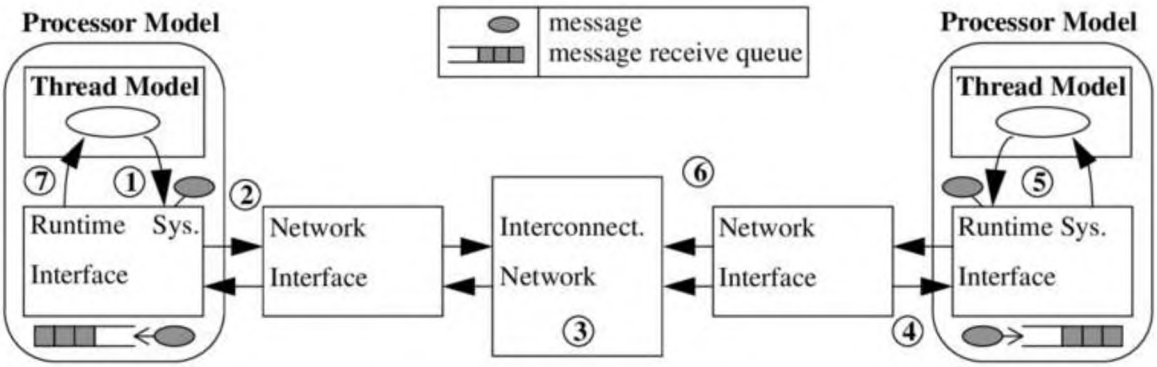


Рисунок 2.9 – RDA модель

Головна перевага цієї моделі полягає в уніфікації та широкому виборі інструментів для розробки додатків. Проте є й недоліки:

– завантаження мережі стає значною через нерозподіленість програми під час обміну даними між клієнтом і сервером;

– виникнення адміністративних проблем.

У DBS-моделі комп'ютер-клієнт відповідає лише за функції представлення. На сервері розміщується база даних, яка містить процедури баз даних, де здійснюються прикладні функції. Також на сервері розташовано ядро, що

контролює доступ до даних (рис. 2.10).

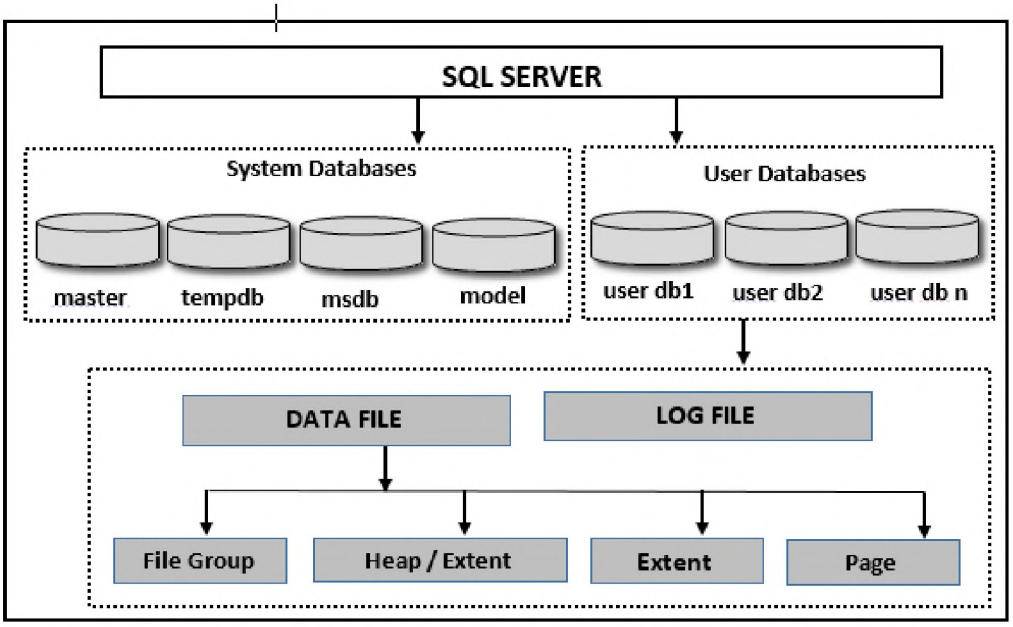


Рисунок 2.10 – DBS модель

Переваги моделі DBS включають:

– централізоване адміністрування бізнес-функцій.

– зниження обсягу мережевого трафіку.

– розподіл процедур між кількома додатками.

– ефективне використання ресурсів комп'ютера (використовується один раз створений план виконання процедур).

Недоліки DBS включають:

– реалізація збережених процедур через процедурні розширення SQL.

– обмежена мобільність системи щодо СУБД (систем управління базами даних).

– відсутність засобів для налагодження та тестування.

– неефективне використання обчислювальних ресурсів.

– потреба у різноманітних методах взаємодії між клієнтом та сервером для децентралізації додатків.

Зазвичай використовують комбіновані методи реалізації, де підтримка

цілісності даних та виконання простих прикладних завдань відводяться збереженим процедурам, а виконання складних завдань доручається комп'ютеру-клієнту з встановленим спеціальним програмним забезпеченням.

2.4 Управління розподіленими ресурсами

Одна з ключових відмінностей між розподіленими та централізованими системами полягає в організації зв'язків між процесорами. Основу для спілкування між процесорами утворює передача повідомлень через мережу. У найпростішому випадку це складається з двох операцій: відправлення та отримання повідомлень. Ефективність мережі залежить від того, який підхід обрано для цих операцій. На вибір впливають спосіб призначення адреси, характеристики викликів (блокуючі/неблокуючі), надійність протоколу обміну повідомленнями та методи буферизації.

– методи адресації впливають на надсилання повідомлень; у простих мережах застосовують прості методи, де адреса має стале значення; але для складних систем такий підхід не ефективний, тому застосовують більш складні методи:

– використання фізичних адрес мережевих адаптерів ефективне у випадках, коли кожен процес працює на окремій машині;

– використання імен призначення, номерів машин та процесів має свої обмеження, так як жорстко прив'язує до певних машин;

– централізований механізм розподілу адрес процесів стикається з обмеженнями у розширенні системи;

– дозвіл процесам вибирати власні ідентифікатори з великого адресного простору має високу потенційну ймовірність конфліктів імен, але забезпечує гнучкість системи;

– використання серверів імен забезпечує використання символьних імен високого рівня, що завдання окремої машини;

– застосування спеціальної апаратури, такої як мережні адаптери, забезпечує унікальні номери процесів.

Блокуючі та неблокуючі базиси (синхронні та асинхронні) визначають поведінку системи. Процес, який очікує відпрацювання блокуючого примітиву, зупиняється до завершення цієї операції. У випадку неблокуючих примітивів, процес отримує керування без очікування завершення операції. Основною перевагою неблокуючих примітивів є підвищена продуктивність, оскільки вони працюють паралельно, але їх складність у створенні програм може бути недоліком.

Надійні та ненадійні базиси стосуються доставки повідомлень. Використання блокуючого елементу може призводити до втрати повідомлень, оскільки процес чекає на момент відправлення, не маючи підтвердження про доставку. Це проблему можна вирішити трьома різними методами:

– система, що не несе зобов'язань щодо доставки, покладає на користувача відповідальність за надійність взаємодії;

– ядро машини, що отримує повідомлення, надсилає підтвердження отримання, яке розблоковує процес користувача; це застосовується для кожного повідомлення.

– у випадках, де запиту слідує відповідь, доцільно використовувати саму відповідь як підтвердження доставки, заблоковуючи процес до отримання відповіді, а при відсутності відповіді протягом тривалого часу, система надсилає запит службі, що запобігає втратам запитів.

Вибір між блокуючими та неблокуючими примітивами впливає на продуктивність системи, а обрані методи доставки повідомлень визначають рівень надійності взаємодії.

Висновки до розділу 2

У розділі розглянуто особливості апаратних та програмних компонент систем розподілених обчислень. Підкреслено способи організації процесорів у розподілених системах та їх архітектурні особливості. Визначено різницю між мультипроцесорами та мультикомп'ютерами, за ознакою спільонго використання оперативної пам‘яті. Розглянуті критерії класифікації багатопроцесорних систем та їх характеристики, визначені класи: симетричні мультипроцесорні системи, системи з масовим паралелізмом та кластери.

Визначено два основних типи мережевих операційних систем: однорангові та дворангові, які відрізняються у розподілі функцій між комп'ютерами у мережі. Мережеві операційні системи мають простішу структуру порівняно з розподіленими, проте вони не досягають повної прозорості, яка характерна для розподілених ОС. Недоліками мережевих операційних систем є складність у використанні та управлінні, хоча вони надають можливість легко додавати, замінювати чи видаляти нові комп'ютери завдяки їхній незалежності у мережі.

Розподілені системи використовують проміжне програмне забезпечення (middleware), яке комбінує переваги обох типів операційних систем, забезпечуючи прозорість розподілу та ефективне масштабування, та це програмне забезпечення слугує для абстрагування програм від базових платформ та забезпечення відповідної моделі програмування для розробників.

РОЗДІЛ 3
НАЛАШТУВАННЯ РЕАЛІЗАЦІЇ ВІРТУАЛІЗОВАНОЇ РОЗПОДІЛЕНОЇ ОБЧИСЛЮВАЛЬНОЇ СИСТЕМИ

3.1 Створення віртуальних машин

Віртуальна машина – програма, яка емулює реальний (фізичний) комп'ютер з усіма його компонентами (жорсткий диск, привід, BIOS, мережеві адаптери тощо). На такий віртуальний комп'ютер можна встановити операційну систему, драйвери, програми тощо.

Створюємо дві віртуальні машини. Вибираємо пункт Launch Instance (рис. 3.1).

Рисунок 3.1 – Створення віртуальної машини

Переходимо до розділу Community AMIs (рис. 3.2).

Рисунок 3.2 – Інтерфейс налаштувань віртуальної машини Community AMIs

Для створення режиму файлу дампу SQL не потрібно нічого, окрім вказання цільового розташування, де ви хочете зберегти файл SQL. Після завершення переносу рекомендується ретельно перевірити перенос бази даних, включаючи всі включені таблиці з індексами, щоб уникнути будь-яких конфліктів у майбутньому. Після перенесення БД далі необхідно провести додаткові налаштування, зокрема для прав доступу.

Вибираємо образ ami-6d7f2f28, що містить CentOS_5.4_x64. Залежно від очікуваного навантаження вибираємо апаратну конфігурацію віртуальної машини. В даному випадку вибираємо конфігурацію m1.large: 2 ядра приблизно по 2 GHz і 7.5GB оперативної пам'яті (рис. 3.3).

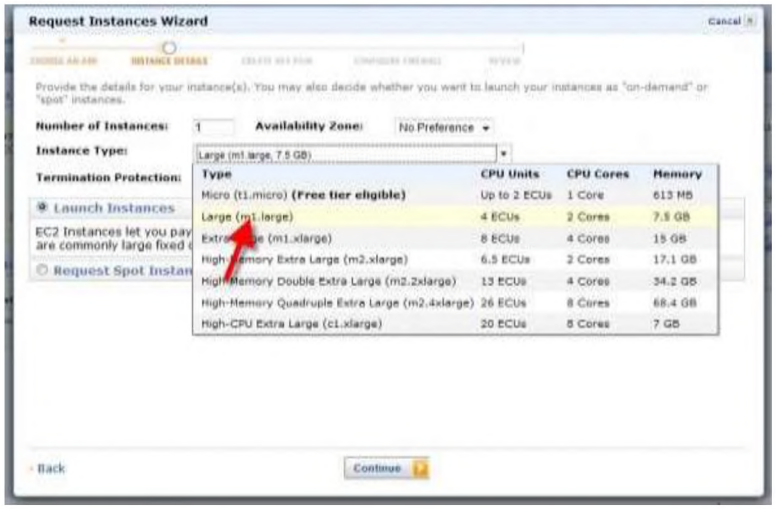


Рисунок 3.3 – Інтерфейс вибору конфігурації m1.large віртуальної машини

Ядро Linux і initrd залишаємо за замовчуванням (рис. 3.4).

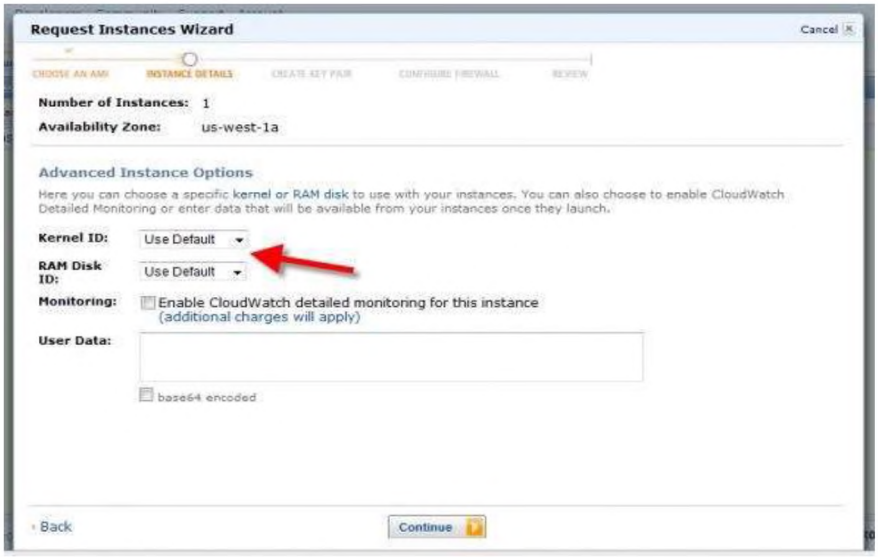


Рисунок 3.4 – Інтерфейс вибору ядра віртуальної машини

Даємо машині зрозумілу назву, що дозволяє відрізнити її від інших машин кластера (рис. 3.5).

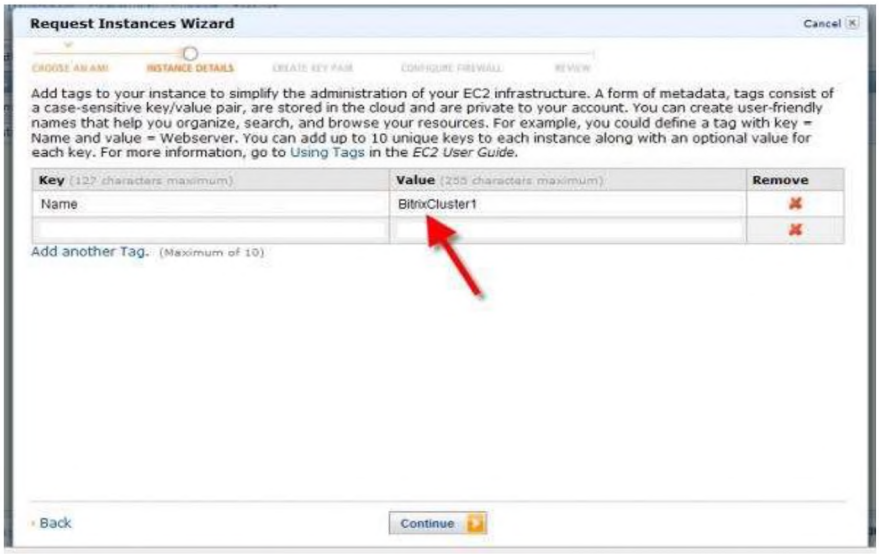


Рисунок 3.5 – Надання імені віртуальної машини

Створюємо або вибираємо зроблену раніше пару ключів (для подальшої авторизації по SSH) (рис. 3.6).

Рисунок 3.6 – Створення ключів шифрування

Створений закритий ключ необхідно імпортувати до PuTTYgen та перезберегти у формат закритого ключа Putty (рис. 3.7). Оскільки доступ на віртуальні машини здійснюється за замовчуванням із використанням закритого ключа, рекомендується задати пароль для його захисту.

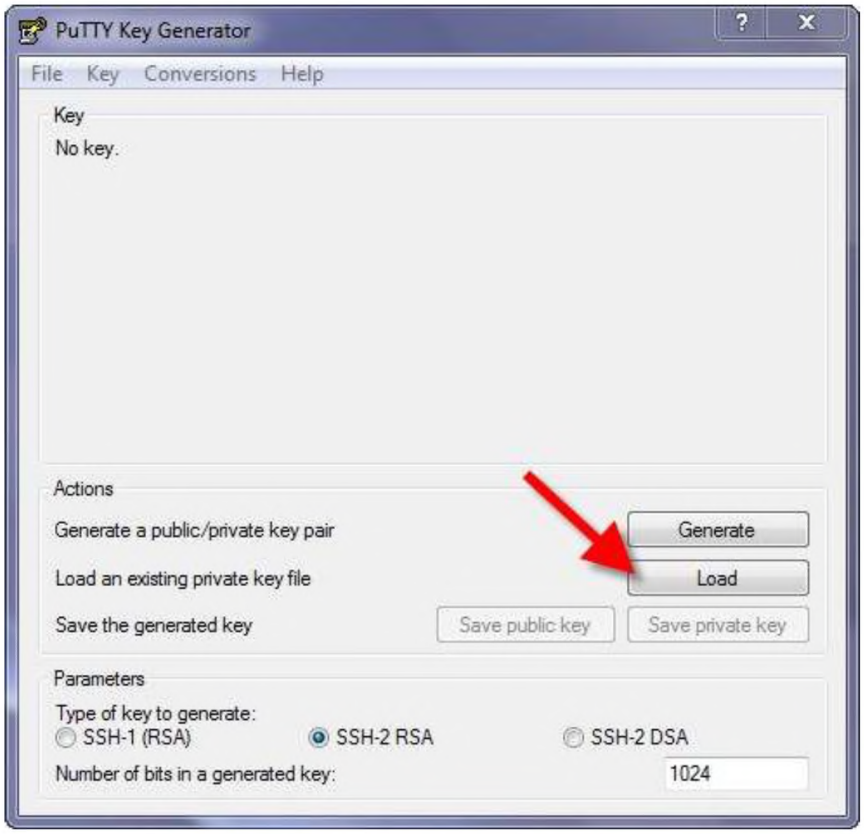


Рисунок 3.7 – Імпорт ключа шифрування до PuTTYgen

Створюємо або вибираємо створену раніше групу безпеки з наступними параметрами (рис. 3.8):

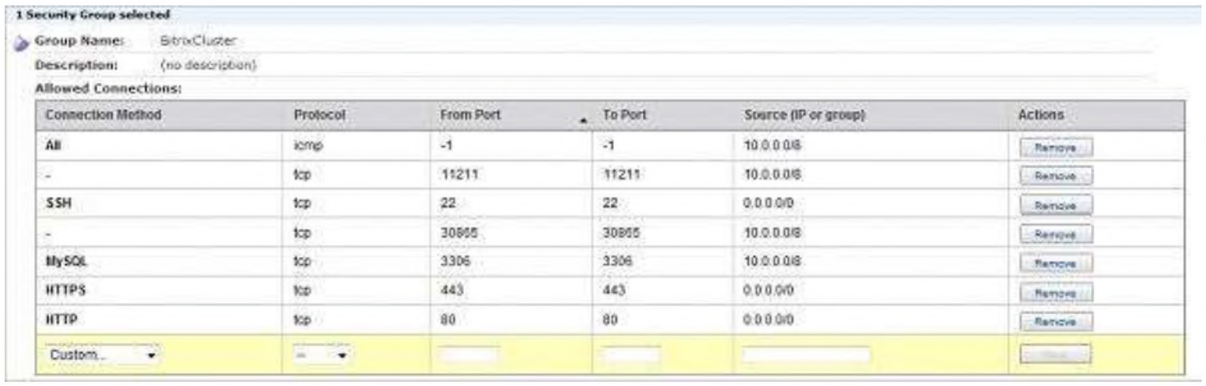


Рисунок 3.8 – Налаштування групи безпеки

Порт 11211 використовується memcache, 30865 для csync2. Після налаштування кластера необхідно обов'язково закрити доступ ззовні кластера до портів memcached. Аналогічно запускаємо другу віртуальну машину (рис. 3.9).

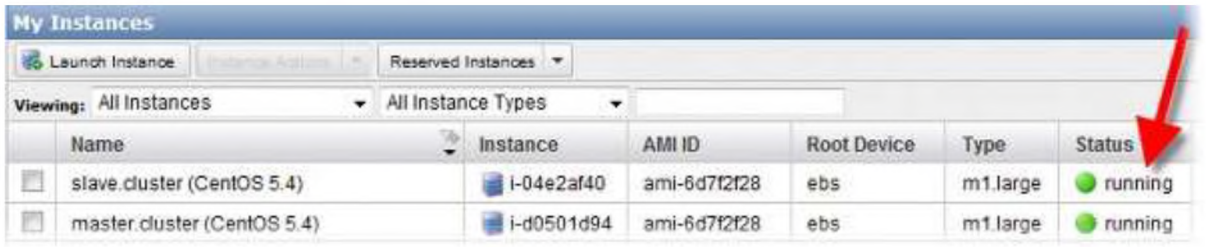


Рисунок 3.9 – Запуск другої віртуальної машини

Отже, запущено дві віртуальні машини необхідної продуктивності із встановленою 64-розрядною операційною системою CentOS. Для встановлення та налаштування вебплатформи ми використовували безкоштовний пакет «Веботочення». Цей пакет призначений для швидкої та простої установки та конфігурації всього стека ПЗ.

Як показують тести, приріст продуктивності на сервері, на якому встановлено веботочення, порівняно з «голим» сервером із налаштуваннями за замовчуванням – як мінімум, у 2-3 рази вище. Налаштування веботочення збалансовані під великі навантаження.

3.2 Налаштування реплікації MySQL, аварійне перемикання slave-master

Реплікація (від латів. replico – повторюю) – це тиражування змін даних з головного сервера БД на одному або кількох залежних серверах. Головний сервер – майстер, а залежні – репліки (рис. 3.10).

Реплікація проводиться з допомогою бінарних логів, які ведуть майстра. Вони зберігаються всі запити, які призводять (чи потенційно які призводять) до змін БД. Бінарні логи передаються на репліки та збережені запити виконуються, починаючи з певної позиції. Важливо розуміти, що з реплікації передаються не самі змінені дані, лише запити, викликають зміни.

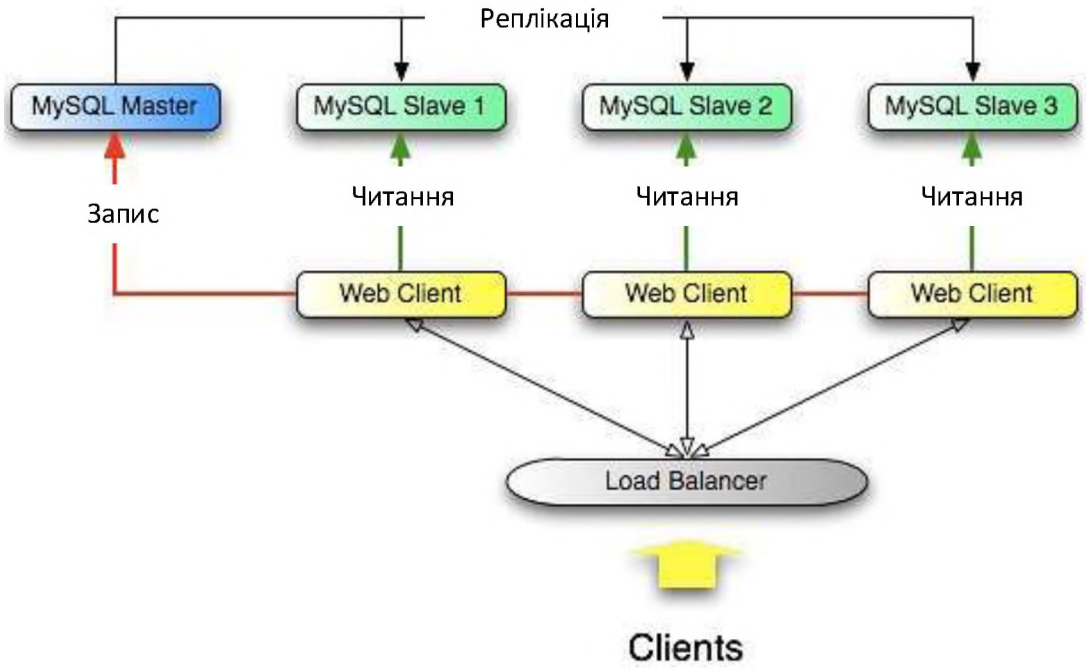


Рисунок 3.10 – Пояснення механізму реплікації

Реплікація MySQL використовується для вирішення наступних завдань:

1. Збільшення продуктивності СУБД шляхом підключення до неї серверів для адаптації до зростання навантаження. Продуктивність зростає за рахунок виділення одного сервера переважно для модифікації даних (master) та інших для

читання даних (slaves). Це рішення особливо ефективне для вебдодатків: у даної категорії додатків більшість запитів до СУБД – запити на читання.

2. Онлайн бекап – дані передаються в режимі онлайн на резервні slave-сервера. При відмові основного сервера одному з резервних slave-серверів є «свіжа» копія даних.

3. Організація ефективного резервного копіювання: бекап виготовляється з резервного сервера без переривання / уповільнення роботи основного сервера СУБД. При необхідності для здійснення цілісного бінарного бекапу СУБД (особливо InnoDB) можна зупинити резервний сервер на необхідний час, виконати бекап і запустити резервний сервер знову.

4. Забезпечення високої доступності: при виході з експлуатації одного із серверів СУБД система продовжує обробляти запити.

Для забезпечення максимальної надійності реплікації рекомендується встановити параметри MySQL таким чином:

innodb_flush_log_at_trx_commit = 1

sync_binlog = 1

sync_relay_log = 1

sync_relay_log_info = 1

Встановлення таких параметрів може призвести до загального зниження продуктивності системи. Для підвищення продуктивності можна використовувати такі параметри (загрожує втрата даних кількох транзакцій у момент аварії на базі даних):

innodb_flush_log_at_trx_commit = 2

sync_binlog = 0

Якщо у сервера, що настроюється, динамічна IP-адреса/hostname, рекомендується явно задати параметри: relay-log, relay-log-index.

Для роботи реплікації облікові записи основного та резервних master/slave-серверів повинні мати крім стандартних також привілеї: SUPER, REPLICATION CLIENT, REPLICATION SLAVE. Їх три, але для простоти адміністрування рекомендується поєднати в одну.

Обліковий запис для роботи програми з базою даних (БД) для всіх нод кластера визначається в dbconn.php.

Під цим же обліковим записом (саме логін/пароль) при додаванні slave-бази через адміністративний інтерфейс система намагається запустити потоки slave (CHANGE MASTER TO #логін/пароль облікового запису в masterБД#), тому їй також потрібні привілеї: REPLICATION SLAVE.

Обліковий запис для управління slave-нодами з адміністративного інтерфейсу вебкластера створюється у кожній slave-БД і вказується при підключенні slave-ноди в адміністративному інтерфейсі вебкластера.

Обліковий запис для завантаження бінарних логів у slave-БД з master-БД (для власне організації процесу реплікації) створюється на master-БД для кожного з'єднання slave-БД (можна використовувати один обліковий запис для всіх слейвів).

Якщо сервери СУБД кластера розташовані у різних дата-центрах, необхідно налаштувати на них єдину часову зону. При додаванні резервного/slave-сервера СУБД майстер налаштування реплікації перевіряє всі необхідні параметри (рис. 3.11).

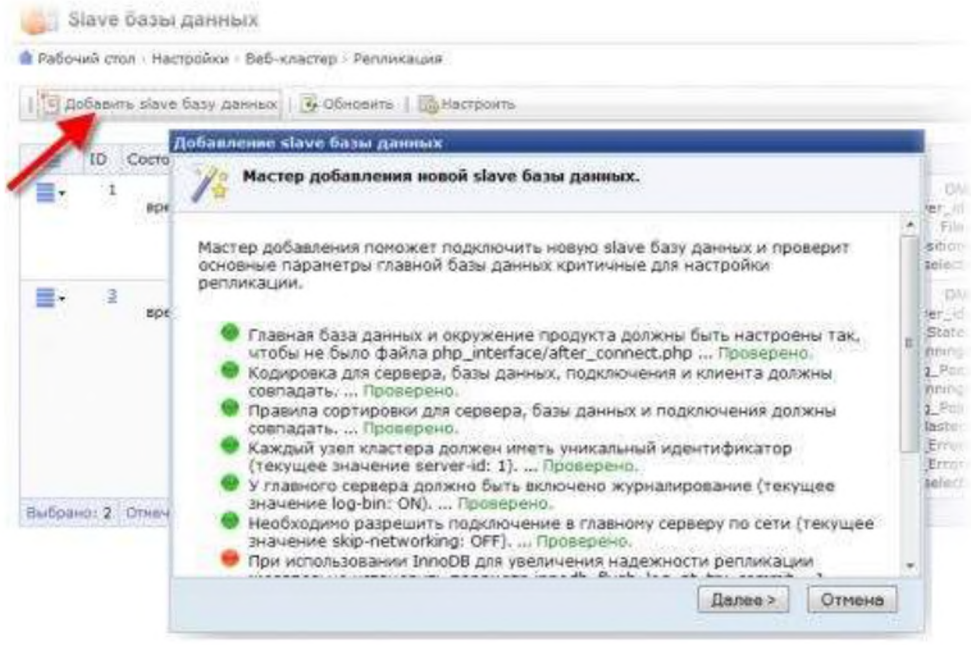


Рисунок 3.11 – Майстер додавання нової бази даних

Після успішного додавання slave-сервера відображається його статус, як показано на рис. 3.12.

Рисунок 3.12 – Статус slave-сервера

Реплікація після налаштування працює надійно та потребує мінімального адміністрування. Проте рекомендується періодично перевіряти її стан утилітами моніторингу операційної системи (nagios, zabbix, monit, linux-ha).

У малоймовірному випадку виникнення помилки на slave-сервері рекомендується його переініціалізувати – заново залити на нього дані з основного сервера. Для цього потрібно його Припинити використовувати, а потім Почати використовувати у розділі Реплікація (Налаштування > Вебкластер > Реплікація).

Можна вільно зупиняти slave-сервер, у т.ч. для здійснення логічного та цілісного бінарного резервного копіювання засобами MySQL та операційної системи. У цьому не переривається робота основного сервера СУБД.

У разі відмови основного (master) сервера СУБД, необхідно вручну або автоматично переключити скриптом кластер на інший master-сервер СУБД. Для цього зазвичай slave-сервер, що зберігає останні репліковані дані, переводять у основний режим.

Загальна схема цієї процедури така:

1. Закрити доступ клієнтів до вебпрограми.

Якщо використовується дворівнева конфігурація (фронтенд nginx – бекенд apache і т.п.), рекомендується на фронтенді відключити доступ до бекенду

(додатку) і віддавати при зверненні клієнтів до кластера інформаційну сторінку про регламентні роботи.

2. Зупинити на всіх slave-серверах потік отримання оновлень бінарного лога з основного (master) сервера:

STOP SLAVE IO_THREAD;

SHOW PROCESSLIST;

Від потоку виконання команд slave (SQL_THREAD) з'явиться повідомлення: «Has read all relay log; Відразу зупиняти slave командою STOP SLAVE не рекомендується, тому що. не всі SQL-команди можуть бути виконані з relay-лога (через відставання і т.п.), а при перемиканні slave на новий майстер, relay-лог буде очищений і, можливо, загубиться частина даних.

3. Підготувати новий master-сервер.

Перевірка ведення бінарного лога на slave-сервері, який ми хочемо зробити master-сервером:

SHOW VARIABLES LIKE 'log_bin'

log_bin | ON

Перевірте, що запити з master не логуються:

SHOW VARIABLES LIKE 'log_slave_updates';

log_slave_updates | OFF

Повна зупинка slave – потоки читання бінарного лога та виконання SQL-команд:

STOP SLAVE;

RESET MASTER;

Команда RESET MASTER необхідна для очищення бінарного лога нового master, інакше, якщо в бінарному лозі будуть записи (застарілі тощо), вони програються на slave-серверах, що підключаються до нього.

4. Переключити slave-сервер на новий master-сервер.

На всіх slave-серверах виконується:

STOP SLAVE;

CHANGE MASTER TO MASTER_HOST='#new_master_host_name#';

START SLAVE;

У момент виконання на slave CHANGE MASTER… очищається relay лог slave-сервера, а позиція, з якої читається бінарний лог master сервера, встановлюється, якщо не задано інше, значення за замовчуванням: перший файл бінарного лога, 4 позиція (тобто в саме початок бінарного лога master-сервера).

5. Переключити вебдодаток на новий master-сервер.

Для налаштування кластера на використання нового master-сервера його характеристики ($DBHost) вписуються у файл php_interface/dbconn.php Якщо інтервал синхронізації статичного контенту кластера досить великий, вручну запускається процес синхронізації на master-сервері.

6. Відкрити доступ клієнтів до вебпрограми.

7. Підключення master-сервера, що вийшов з ладу, як slave-сервера.

В адміністративному розділі на сторінці Реплікація (Налаштування > Вебкластер > Реплікація) «Додати slave базу даних». Відбувається переініціалізація slave-сервера: до нього переноситься поточна база даних з master-сервера та включається реплікація.

3.3 Кластеризація вебсервера

Використання декількох вебсерверів для роботи сайту означає, що будь-який запит будь-якого відвідувача може потрапити на будь-який з вузлів кластера.

Отже, на кожному вебсервері має бути однаковий контент (файли); сесія користувача повинна бути «прозорою» для всіх серверів вебкластера.

Існує два можливі методи забезпечення синхронізації даних на серверах:

- використання загального дискового сховища даних;
- використання інструментів синхронізації даних між локальними сховищами різних серверів.

На кожному сервері необхідно налаштувати сторінку, на якій відображатиметься статистика вебсервера Apache (за допомогою модуля mod_status). Для цього необхідно:

1. Внести зміни до конфігураційного файлу Apache (/etc/httpd/conf/httpd.conf) – додати:

<IfModule mod_status.c>

ExtendedStatus On

<Location /server-status>

SetHandler server-status

Order allow,deny

Allow from 10.0.0.1

Allow from 10.0.0.2

Deny from All

</Location>

</IfModule>

Директива Location означає адресу, за якою буде доступна статистика, рядки Allow from визначають, з яких IP-адрес статистика буде доступна для перегляду. Можна вказати IP всіх вебсерверів кластера.

Перечитати конфігураційні файли Apache за допомогою команди:

service httpd reload

2. Внести зміни до конфігураційного файлу nginx (/etc/nginx/nginx.conf)

- у першій секції server додати:

Location ~ ^/server-status$

{proxy_pass http://127.0.0.1:8888; }

Перечитати конфігураційні файли nginx за допомогою команди:

service nginx reload

3. Відредагувати файл /www/.htaccess, додавши після рядка: RewriteCond %{REQUEST_FILENAME} !/bitrix/urlrewrite.php$ рядок:

RewriteCond %{REQUEST_URI} !/server-status$

Після внесення всіх необхідних змін адресу server-status можна додати конфігурацію кластера у вікнах інтерфейсу, як показано на рис. 3.13 – рис. 3.15.

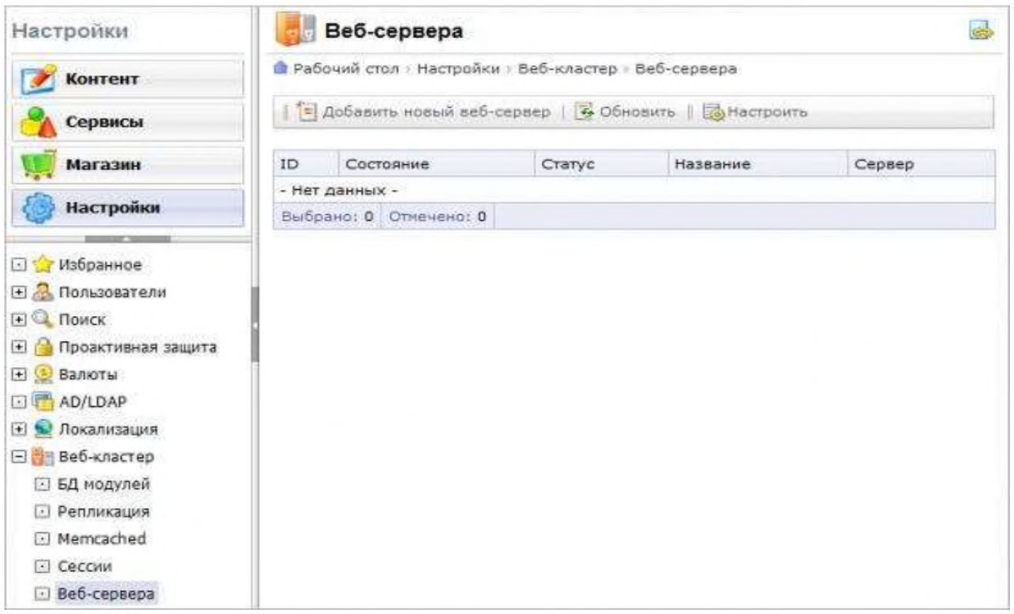


Рисунок 3.13 – Інтерфейс налаштування веб кластера

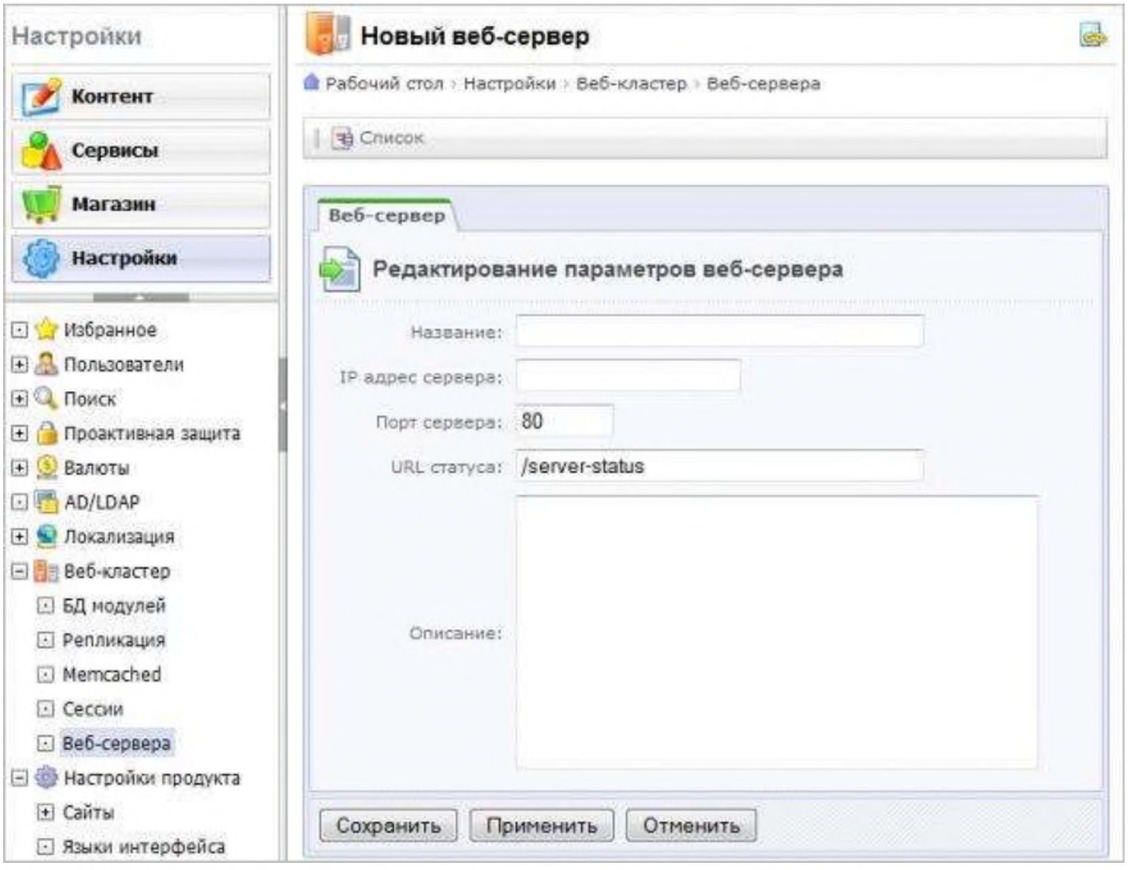
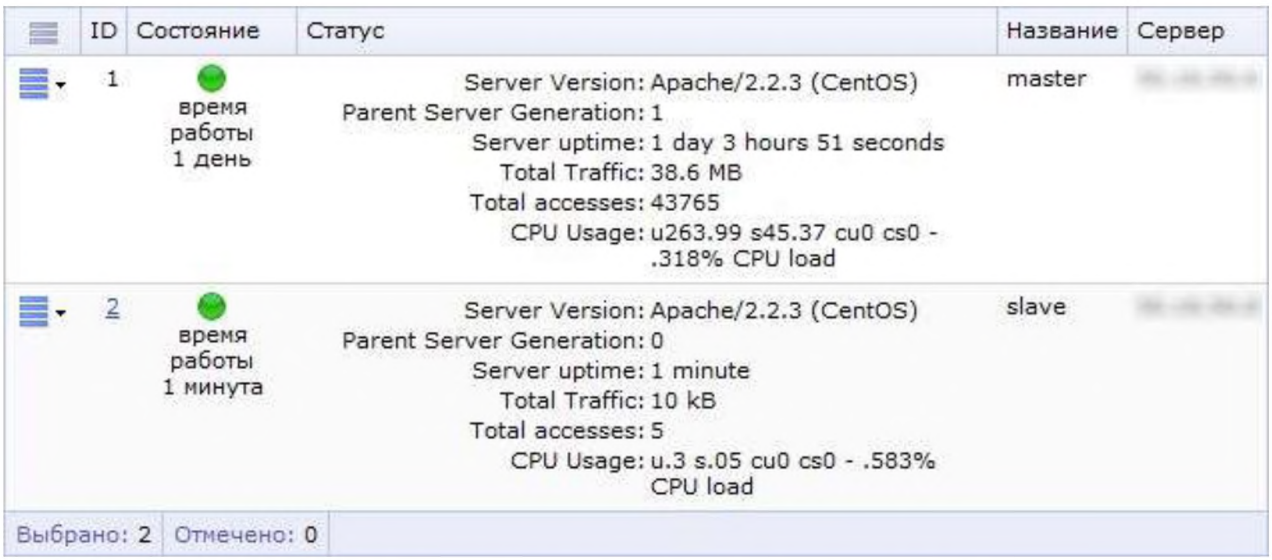


Рисунок 3.14 – Редагування параметрів вебсервера у кластері



ID	Состояние	Статус	Название	Сервер
1	время работы 1 день	Server Version: Apache/2.2.3 (CentOS) Parent Server Generation: 1 Server uptime: 1 day 3 hours 51 seconds Total Traffic: 38.6 MB Total accesses: 43765 CPU Usage: u263.99 s45.37 cu0 cs0 - .318% CPU load	master	
2	время работы 1 минута	Server Version: Apache/2.2.3 (CentOS) Parent Server Generation: 0 Server uptime: 1 minute Total Traffic: 10 kB Total accesses: 5 CPU Usage: u.3 s.05 cu0 cs0 - .583% CPU load	slave	

Выбрано: 2 Отмечено: 0

Рисунок 3.15 – Статус вебсервера у кластері

Після успішного створення вебкластеру, статус його серверів можна проконтролювати, як показано на рис. 3.15.

3.4 Синхронізація даних між серверами

Завдання організації спільного файлового сховища даних вебкластера можна вирішити різними, добре відомими способами, починаючи від «розшарування» папки з файлами продукту на одній із нод, закінчуючи високопродуктивними SAN/NAS-рішеннями рівня підприємства.

Якщо вебкластер створюється на двох нодах, на одній ноді можна запустити сервер NFS/CIFS і звертатися до файлів, що зберігаються на ньому, з інших нод вебкластера. Для збільшення продуктивності можна організувати виділений NFS/CIFS сервер, оптимізований виключно під файлові операції, використовуваний усіма нодами вебкластера.

Інший підхід в організації декількох серверів полягає у використанні локальних сховищ (дисків) та постійної синхронізації файлів між ними. Існує багаточ інструментів, що дозволяють вирішити це завдання, починаючи з простих

утиліт копіювання файлів (ftp, scp) і закінчуючи спеціалізованим програмним забезпеченням для синхронізації даних між серверами (rsync, unison).

У кластері для синхронізації даних між серверами ми будемо використовувати утиліту csync2. Переваги csync2:

- висока швидкість роботи;
- низьке споживання ресурсів (CPU, дискові операції);
- простота налаштування для обміну даними між будь-якою кількістю серверів;
- можливість синхронізації видалення файлів;
- захищений обмін даними між хостами (SSL).

У деяких дистрибутивах Linux (наприклад, в Ubuntu) csync2 вже є в репозиторіях і доступний для встановлення з пакетів. Один з можливих варіантів встановлення:

1. Встановлюємо необхідні доступні пакети (бібліотеки rsync використовуються в csync2, з xinetd запускатиметься серверна частина csync2):

yum install rsync librsync-devel xinetd

2. Встановлюємо libtasn1 – бібліотеку для роботи з X.509 (потрібна для csync2):

wget http://ftp.freshrpms.net/redhat/testing/EL5/cluster/x86_64/libtasn1-1.1-1.el5/libtasn1-1.1-1.el5.x86_64.rpm

wget http://ftp.freshrpms.net/redhat/testing/EL5/cluster/x86_64/libtasn1-1.1-1.el5/libtasn1-devel-1.1-1.el5.x86_64.rpm

wget http://ftp.freshrpms.net/redhat/testing/EL5/cluster/x86_64/libtasn1-1.1-1.el5/libtasn1-tools-1.1-1.el5.x86_64.rpm

rpm -ihv libtasn1-1.1-1.el5.x86_64.rpm libtasn1-devel-1.1-1.el5.x86_64.rpm

3. Встановлюємо sqlite:

wget http://ftp.freshrpms.net/redhat/testing/EL5/cluster/x86_64/sqlite2-2.8.17-1.el5/sqlite2-2.8.17-1.el5.x86_64.rpm

wget http://ftp.freshrpms.net/redhat/testing/EL5/cluster/x86_64/sqlite2-2.8.17-1.el5/sqlite2-devel-2.8.17-1.el5.x86_64.rpm

rpm -ihv sqlite2-2.8.17-1.el5.x86_64.rpm sqlite2-devel-2.8.17-1.el5.x86_64.rpm

4. Встановлюємо безпосередньо csync2:

wget http://ftp.freshrpms.net/redhat/testing/EL5/cluster/x86_64/csync2-1.34-4.el5/csync2-1.34-4.el5.x86_64.rpm

rpm -ihv csync2-1.34-4.el5.x86_64.rpm

5. Після встановлення вже доступні згенеровані SSL-сертифікати, проте, якщо їх потрібно знову згенерувати, зробити це можна так:

openssl genrsa -out /etc/csync2/csync2_ssl_key.pem 1024

openssl req -new -key /etc/csync2/csync2_ssl_key.pem -out /etc/csync2/csync2_ssl_cert.csr

openssl x509 -req -days 600 -in /etc/csync2/csync2_ssl_cert.csr -signkey /etc/csync2/csync2_ssl_key.pem -out /etc/csync2/csync2_ssl_cert.pem

6. На одному з хостів необхідно згенерувати файл ключів, який потім буде необхідно розмістити на всіх серверах, які братимуть участь у процесі синхронізації:

csync2 -k /etc/csync2/csync2.cluster.key

Файл csync2.cluster.key на всіх машинах повинен бути однаковим і повинен бути розміщений у /etc/csync2/.

Приклад конфігураційного файлу /etc/csync2/csync2.cfg:

group cluster

{host node1.demo-cluster.ru; node2.demo-cluster.us;

key /etc/csync2/csync2.cluster.key;

include /home/ www; exclude /home/bitrix/www/bitrix/php_interface/dbconn.php;

auto younger;

}

де:

- host – адреси серверів, котрим синхронізуються дані;
- key – файл ключів;
- include – директорія, файли якої синхронізуються;
- exclude – винятки (не синхронізувати ці дані);

– auto – спосіб вирішення конфліктів у ситуаціях, коли той самий файл був змінений на декількох серверах одночасно;

– younger – пріоритет має сервер, де були зроблені останні зміни.

У конфігураційному файлі csync2 може бути задано кілька груп синхронізації. csync2 автоматично ігнорує ті групи, в яких не встановлено локальне ім'я машини (яке можна подивитися, наприклад, за допомогою команди hostname). Тому імена машин, дані яких синхронізуються, не повинні змінюватися. Для цього можна використовувати файл /etc/hosts для вказівки постійних IP для певних імен.

Якщо спочатку кожен новий сервер створюється вже з копією даних з інших серверів, можна швидко виконати першу ініціалізацію бази файлів для синхронізації за допомогою команди:

csync2 -cIr

Зробити це потрібно на кожному хості.

обота csync2 на кожному хості складається з двох частин – серверної та клієнтської. Для запуску серверної частини необхідно закоментувати (символом #) у файлі /etc/xinetd.d/csync2 рядок «disable = yes», перезапустити сервіс xinetd та дозволити його запуск під час завантаження системи:

service xinetd restart

chkconfig xinetd on

Клієнтська частина – запуск csync2 з ключем «-x».

Можна визначити (залежно від обсягу даних) необхідну частоту оновлень та прописати запуск csync2, наприклад, через cron.

3.5 Кластеризація кешу (memcached)

Для централізованого та надійного зберігання даних кешу використовується кластер серверів memcached, який активно використовується у таких проектах як Youtube, Facebook, Twitter, Google App Engine.

Для запуску сервера memcached на вебсервері на ОС Linux необхідно виконати команди:

chkconfig memcached on

service memcached start

Для забезпечення максимальної продуктивності вебкластера реалізовано централізоване сховища даних, що розділяється між нодами даних кешу – кеш, створений в результаті ресурсомістких обчислень нодою «А», буде використаний нодою «В» та іншими нодами кластера. Чим більше нод, тим більший ефект дасть використання централізованого кешу.

За рахунок використання в сервері memcached алгоритму LRU в кеші зберігатимуться лише найбільш актуальні дані, а рідко використовувані – витіснятися. Це не дозволить обсягу кешу безмежно розростатися в розмірах (наприклад, через помилки розробників при інтеграції), що призводило не тільки до неефективного використання ресурсів системи, а й негативно позначалося на швидкості роботи з кешем за рахунок зростання його обсягу.

В результаті кеш вебкластера автоматично підтримуватиметься в максимально ефективному стані: як за швидкістю доступу, так і з використанням ресурсів. Рекомендується періодично аналізувати використання кешу додатком та підібрати оптимальний розмір кешу, як показано на рис. 3.16.

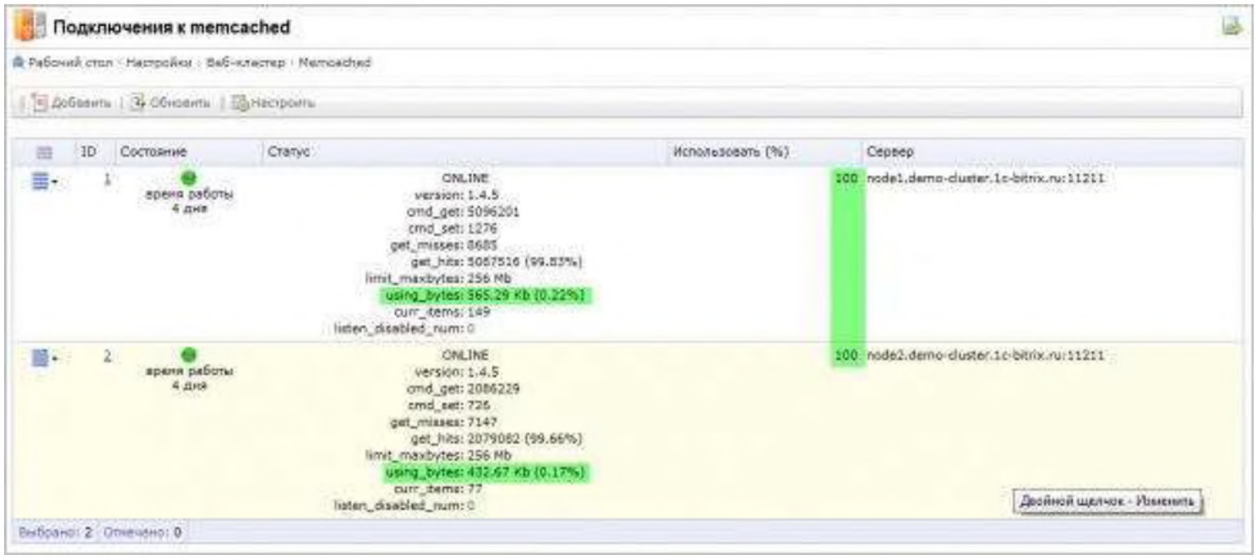


Рисунок 3.16 – Механізм аналізу кешу вебсервера

Якщо сервер memcached має різний розмір (зумовлений, можливо, різним обсягом оперативної пам'яті на фізичних серверах), можна регулювати їх використання через завдання «ваги».

За рахунок кластеризації даних кешу на декількох серверах memcached вихід з ладу одного memcached сервера не виведе з ладу підсистему кешування. У разі збільшення обсягу даних кешу необхідно підключити до вебкластеру додатковий memcached сервер, як показано на рис. 3.17.

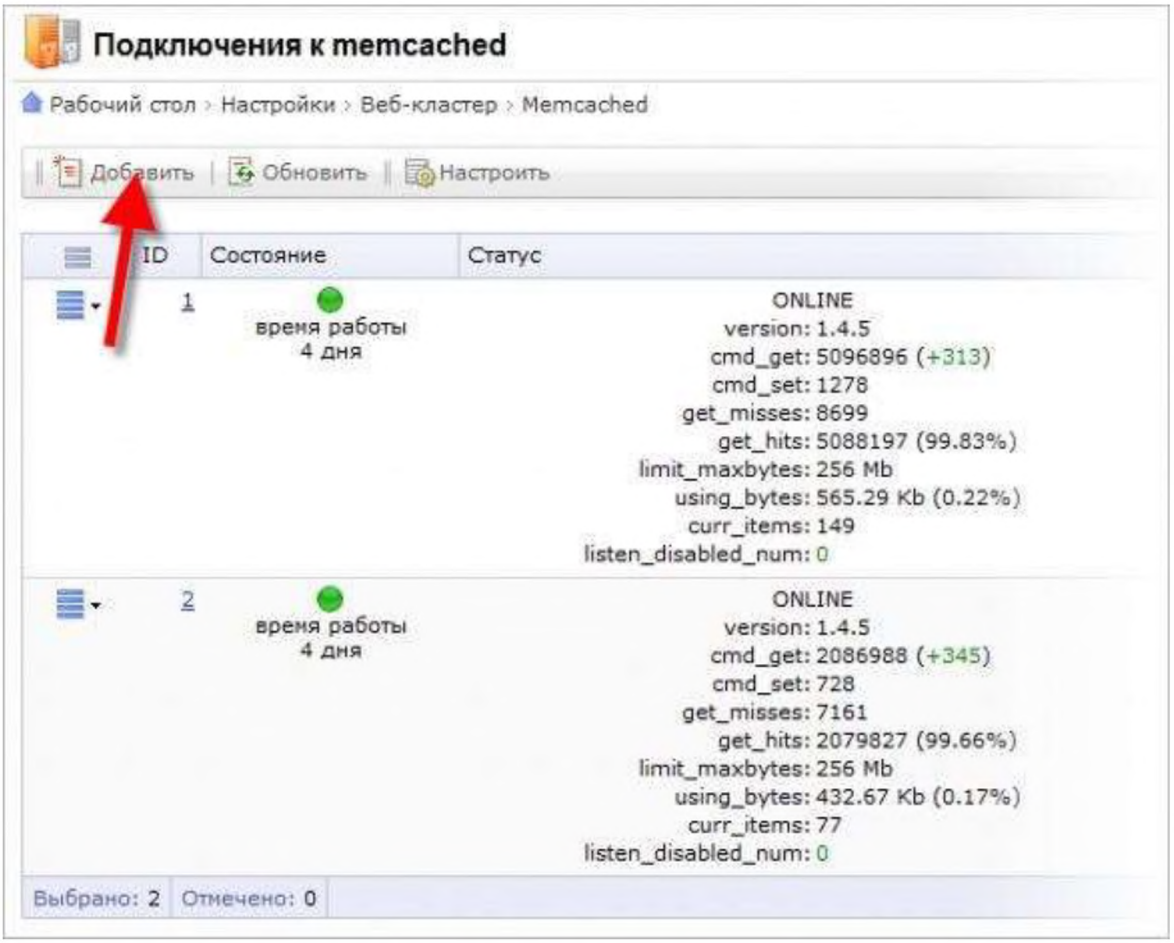


Рисунок 3.17 – Підключення до вебкластеру додаткового memcached серверу

Якщо використовується кілька memcache-серверів, потрібно встановити значення змінної memcache.hash_strategy string у consistent. Таке значення потрібне для наступного: якщо один із memcache-серверів зламається, почнеться ресурсомістка операція перетасовування ключів між memcache-серверами, що

уповільнює роботу вебкластера. А якщо встановити у consistent – проблем практично не буде.

3.6 Спосіб балансування навантаження між нодами вебсервера

Спектр інструментів для розподілу та балансування навантаження між серверами дуже широкий. Існують як дорогі апаратні рішення (наприклад, Cisco CSS, Content Services Switch), так і простіші (але вельми ефективні) програмні засоби.

Балансувальник навантаження AWS використовує простий алгоритм балансування (round robin) та налаштовується досить просто (рис. 3.18). Слід звернути увагу на те, що DNS-ім'я балансувальника створюється автоматично, і необхідно створити для нього простий ідентифікатор CNAME.

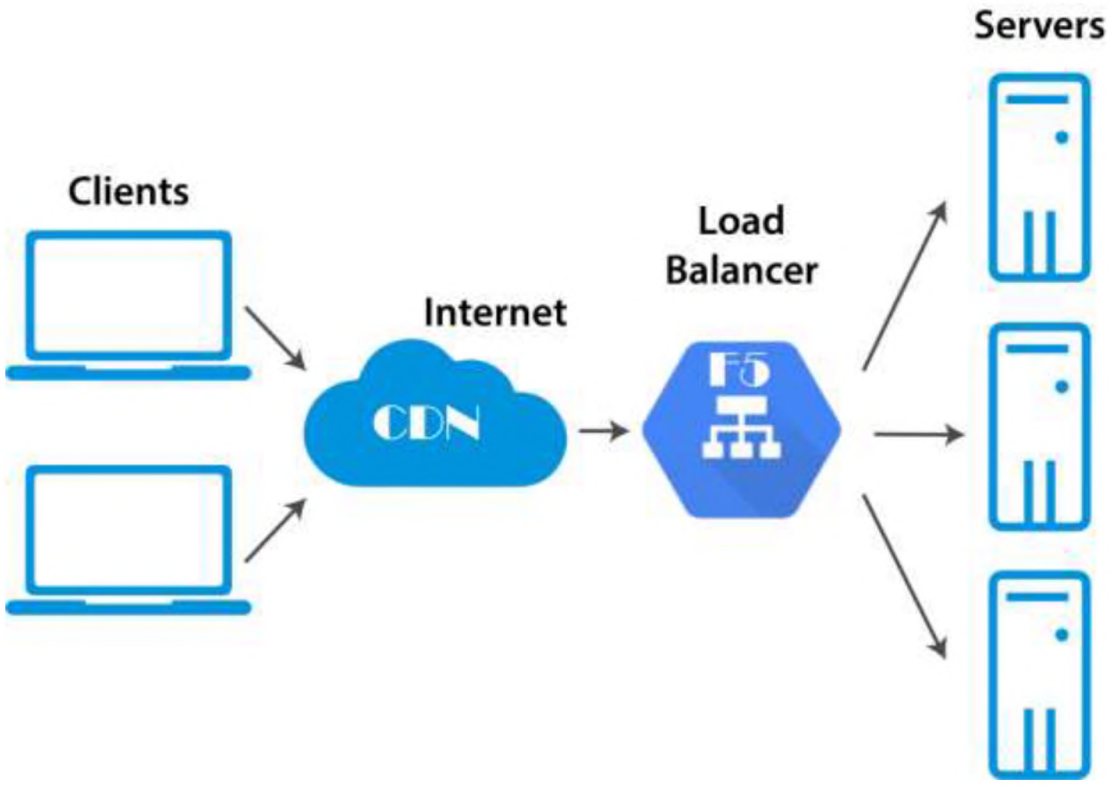


Рисунок 3.18 – Принцип роботи балансувальника навантаження

Балансувальник «слухає» порт 80 і перенаправляє запити на 80 порт нод вебкластера (рис. 3.19).

Рисунок 3.19 – Налаштування балансувальника навантаження вебкластеру

Вказуємо сторінку ноди для перевірки та таймаути, як показано на рис. 3.20.

Рисунок 3.20 – Налаштування ноди балансувальника навантаження

Рекомендується в Ping Path встановити сторінку, звернення до якої не фіксуються в модулі Вебаналітика. Інші параметри вибираються залежно від очікуваного навантаження.

Вибираємо ноди, які будуть за балансувальником (рис. 3.21). У зв'язку з простим алгоритмом балансування (round robin) рекомендується підбирати ноди приблизно однакової продуктивності. Якщо ноди розташовуються у різних дата-центрах (availability zones), рекомендується взяти їх однакову кількість у кожному дата-центрі.

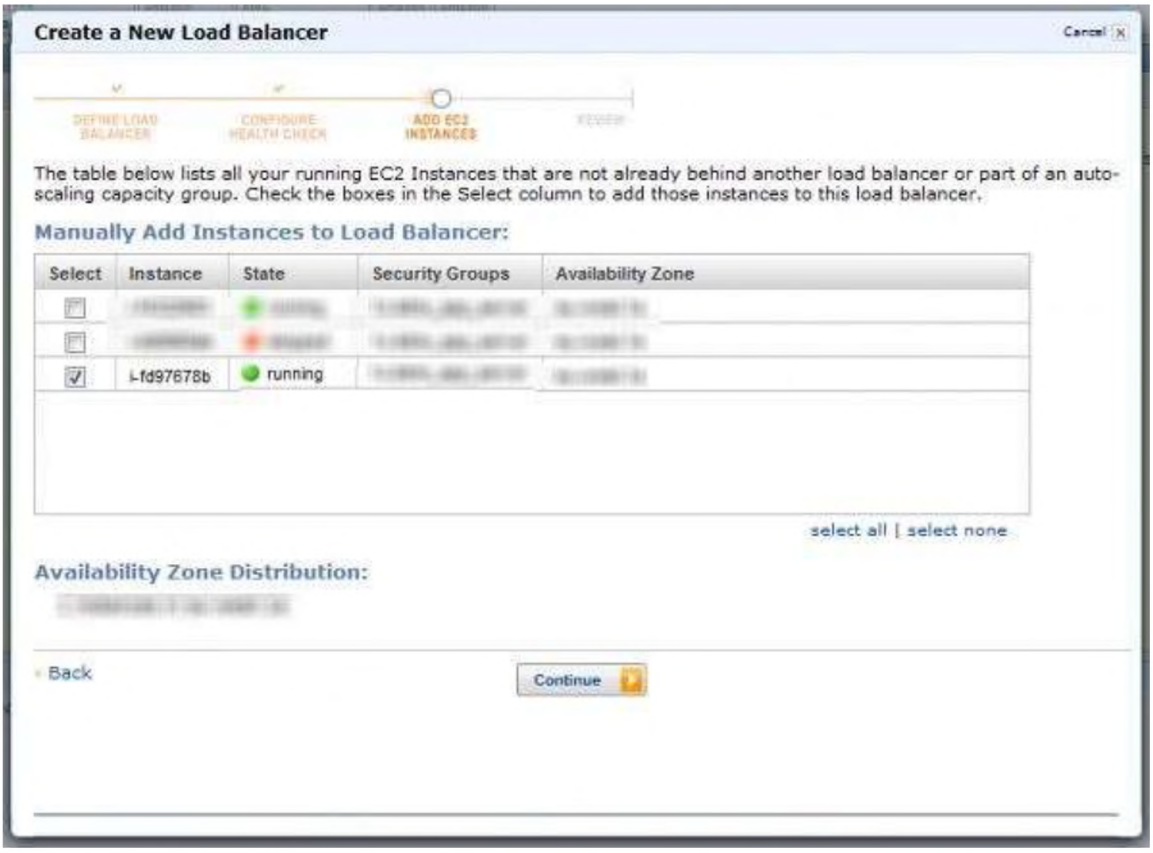


Рисунок 3.21 – Підключення нод до балансувальника

Якщо ми хочемо, щоб усі запити від тих самих клієнтів оброблялися одними і тими ж нодами кластера, необхідно додати прив'язку сесії відвідувача до ноди вебкластера. Для цього в налаштуваннях балансувальника включаємо прив'язку Enable Load Balancer Generated Cookie Stickiness. Тепер, незалежно від того, скільки нод кластера розміщено за балансувальником, вони будуть доступні за єдиним

доменним ім'ям: www.mydomain.com. У разі виходу ноди з ладу/перевантаження балансувальник перестає направляти відвідувачів кластера.

Найпростіший спосіб балансування навантаження та розподілу запитів між вебсерверами – використання механізму round robin DNS. У сервісі Route53 існує ще один простий та зручний механізм фейловеру на базі DNS (Route 53 Health Checks and DNS Failover).

Специфікація стандарту DNS дозволяє вказати кілька різних записів IN A для одного імені. Наприклад: www IN A 10.0.0.1 www IN A 10.0.0.2.

У цьому випадку DNS-сервер у відповідь на запит дозволу імені в IPадресу видаватиме не одну адресу, а весь список:

host www.domain.local www.domain.local has address 10.0.0.1.

www.domain.local has address 10.0.0.2.

При цьому з кожним новим запитом послідовність адрес у списку у відповіді змінюватиметься. Таким чином, клієнтські запити будуть розподілятися між різними адресами і потраплятимуть на різні сервери.

Використання round robin DNS – найпростіший спосіб балансування навантаження, що не вимагає ніяких додаткових засобів, проте він має цілу низку недоліків, тому ми рекомендуємо використовувати його тільки в тому випадку, якщо немає можливості застосувати будь-який інший балансувальник.

Мінуси застосування round robin DNS:

− немає чіткого критерію вибору IP зі списку клієнтом (може вибиратися перший елемент, може кешуватися останнє обране значення, можливі інші варіанти): залежить від реалізації конкретного клієнта;

− немає механізмів визначення доступності вузлів і можливості завдання їх «ваг»: у разі аварії на одному або кількох вузлах кластера навантаження продовжуватиме розподілятися між робітниками і нодами, що вийшли з ладу;

− тривалий час кешування відповідей DNS: у разі аварії та зміні тих чи інших записів у DNS потрібно деякий час (залежить від установок DNS) для оновлення даних на всіх клієнтах.

Розглянемо приклад використання як балансувальник http сервера nginx. Для цього використовується модуль ngx_http_upstream.

При використанні вебкластеру, nginx вже встановлено на його серверах. І для розподілу навантаження можна використовувати безпосередньо один із серверів кластера. Однак для більш простого, гнучкого та зручного конфігурування краще використовувати окремий сервер із встановленим nginx як балансувальник.

У найпростішому прикладі фрагмент конфігураційного файлу nginx (/etc/nginx/nginx.conf), що забезпечує розподіл навантаження між серверами, буде виглядати так (крім стандартних директив, що визначають, наприклад, шляхи та формат log-файлів тощо):

http {upstream backend {server node1.demo-cluster.us; server node2.demo-cluster.us;}

server {

listen 80;

server_name load_balancer;

location / {

proxy_set_header X-Real-IP $remote_addr;

proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; proxy_set_header Host $host: 80;

proxy_pass http://backend; } } }

У секції upstream задаються адреси серверів, між якими розподілятиметься навантаження. У DNS ім'я сайту вказується на той IP, на якому працює сервер із nginx, що розподіляє навантаження. Якщо не вказано жодних додаткових параметрів, запити розподіляються між серверами за принципом round robin. Однак за допомогою модуля ngx_http_upstream можна реалізувати і складніші конфігурації.

У деяких випадках буває зручно передавати всі запити від одного клієнта на той самий сервер, а між серверами розподіляти лише запити різних клієнтів. Для реалізації подібного функціоналу є директива ip_hash. У цьому випадку розподіл запитів ґрунтується на IP-адресах клієнтів.

Якщо для вузлів кластера використовуються сервери різної конфігурації (різної потужності), буває корисно задати «вагу» для тих чи інших хостів, спрямовуючи на них більше або менше запитів.

Для цього слугує параметр weight директиви server.

upstream backend

{ server node1.demo-cluster.us weight=3; server node2.demo-cluster.us;

server node3.demo-cluster.us; }

У такій конфігурації кожні 5 запитів розподілятимуться таким чином:

– 3 – на node1.demo-cluster.us;

– 1 – на node2.demo-cluster.us;

– 1 – на node3.demo-cluster.us.

Примітка: для серверів, які використовують метод розподілу ip_hash, не можна встановити вагу.

Якщо при запиті на сервер виникла помилка, запит буде передано на наступний сервер. Якщо відбулося 3 помилки протягом 30 секунд, то на 30 секунд сервер буде позначений непрацюючим, і протягом цього часу нові запити не надсилатимуться на нього.

3.7 Додавання ноди вебкластера

У зв'язку з навантаженням, що росте, необхідно буде додавати нові ноди до вебкластеру. Фактично необхідно запускати новий фізичний/віртуальний сервер і прописати його в налаштуваннях вебкластера.

Оскільки наш демо-кластер ми запускали в «хмарі» Amazon, ми опишемо послідовність дій саме для AWS. Однак і для будь-якого іншого середовища (будь то VPS або кілька виділених серверів) загальна схема буде приблизно такою самою.

На хмарному хостингу AWS необхідно:

1. Створити диск з даними програми однієї з нод.

2. Створити зі снепшота новий диск, який зберігатиме дані оригінальної ноди.

3. Запустити віртуальну машину з AMI, аналогічною до оригінальної ноди.

4. Вибрати апаратну конфігурацію для віртуальної машини в залежності від очікуваного навантаження. У найпростішому випадку рекомендується вибрати аналогічну оригінальну ноду.

5. Зупинити нову ноду.

6. Відключити від неї диск.

7. Замість вимкненого диска підключити створений на кроці 2 диск.

8. Запустити нову ноду.

9. Прив'язати при необхідності до нової ноди еластичну IP-адресу.

10. Налаштувати нову ноду: якщо використовується csync2, то прописати його налаштування на всіх нодах вебкластера доменне ім'я нової ноди. Запустити на новій ноді необхідні послуги: memcached, mysql-slave.

11. Рекомендується в ініціалізаційні скрипти нової ноди додати її прив'язку до еластичної IP-адреси. Інакше при зупинці/запуску потрібно буде знову прив'язувати до неї IP-адресу вручну.

12. Тепер із новою нодою синхронізується контент із поточних нод вебкластера. Її необхідно додати до балансувальника навантаження.

Отже, додавання нової ноди до вебкластеру – процес, який вимагає кількох кроків у конфігуруванні та синхронізації, але в кінці створює можливість розподілити навантаження та підвищити стійкість системи.

3.8 Налаштування параметрів безпеки вебкластеру

У зв'язку з тим, що вебкластер використовує додаткові послуги (централізоване кешування, синхронізація) і запускається, як правило, на групі машин, розглянемо особливості забезпечення інформаційної безпеки вебкластера.

Для балансування навантаження та захисту від DDOS-атак рекомендується відкрити для публічного доступу 80 порт балансувальника навантаження, обмеживши зовнішній доступ до http портів машин (нод) вебкластера. Це надійно захистить ноди, заховані за балансувальником, від навантаження (наприклад, що виникло при проведенні рекламної інтернет-кампанії), а також суттєво знизить ефективність DDOS-атак.

Необхідно закрити від публічного доступу сервери memcached (tcp порт 11211), відкривши доступ до них з нод вебкластера. Одне з рішень – налаштувати фаєрвол.

Якщо для синхронізації контенту використовується утиліта csync2, необхідно закрити її послуги від публічного доступу (tcp порт 30865), відкривши доступ до них з нод вебкластера.

Як приклад, налаштування фаєрволу для ноди вебкластера наступні:

- 22 (tcp; ssh) – відкритий для підмережі адміністратора;
- 80 (tcp; http) – відкритий для підмережі вебкластера;
- 443 (tcp; https) – відкритий для підмережі вебкластера;
- 3306 (tcp; mysql) – відкритий для підмережі вебкластера;
- 11211 (tcp; memcached) – відкритий для підмережі вебкластера;• 30865 (tcp; csync2) – відкритий для підмережі вебкластера.

Для балансувальника навантаження:

- 80 (tcp; http) – відкритий всім;
- 443 (tcp; https) – відкритий для всіх.

Забезпечення інформаційної безпеки вебкластера – це важливий аспект, що вимагає уважного налаштування портів та обмежень доступу, забезпечуючи захист від різноманітних атак та ефективну роботу системи. Важливо враховувати, що налаштування захисту портів машин вебкластера є лише однією зі складових системи безпеки. Регулярне оновлення програмного забезпечення, встановлення актуальних патчів та моніторинг діяльності системи також є важливими аспектами забезпечення безпеки. Крім того, впровадження механізмів резервного копіювання

та реагування на інциденти дозволяють створити більш повне й ефективне забезпечення безпеки вебкластера.

3.9 Навантажувальне тестування кластера, аналіз різних сценаріїв

Існує багато утиліт для проведення тестів навантажень вебсистем. Від досить простих (ab, що входить до дистрибутиву Apache, siege, httperf) до потужних інструментів, що дозволяють задавати будь-які сценарії користувача і надають найрізноманітнішу статистичну інформацію (JMeter, tsung, WAPT).

Слід звернути увагу, що для мінімізації впливу тестуючого скрипта на результати тестів рекомендується запускати його не на самих тестованих серверах, а на окремому хості. Також на тестованому сайті необхідно на час проведення тесту вимкнути опцію блокування користувача при великій кількості з'єднань у налаштуваннях модуля Вебаналітика, закладка Налаштування, розділ Обмеження активності. Результати першого тесту – зростання навантаження показані на рис. 3.22.

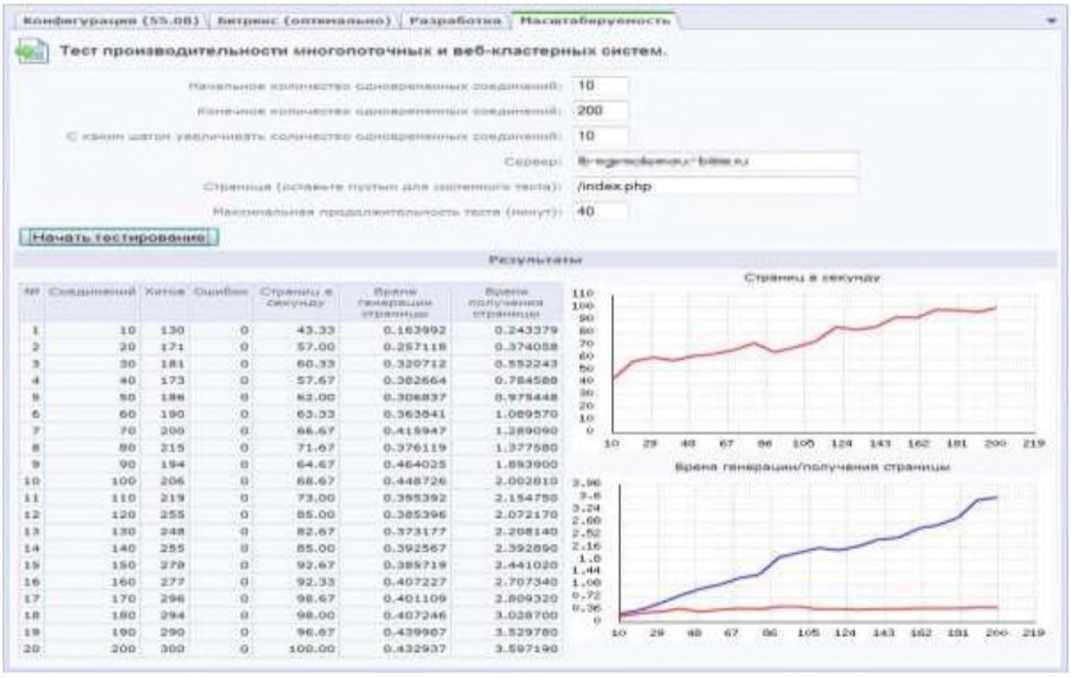


Рисунок 3.22 – Результати тесту зі зростання навантаження

Параметри тестів, що проводяться наступні:

– сервер – у нашому випадку ми даємо тестове навантаження на балансувальник (nginx), який розподіляє навантаження між двома вебсерверами кластера (на кожному сервері працюють і додані в кластер вебсервер, MySQL, memcached);

– сторінка – URL, на яку надсилаються запити;

– початкова кількість одночасних з'єднань, кінцева кількість, крок збільшення з'єднань – параметри навантаження.

У нашому тестовому прикладі ми починаємо надсилати запити на індексну сторінку до 10 одночасних потоків і поступово збільшуємо їхню кількість до 200.

За другим графіком (рис. 3.22) бачимо таку картину:

– вся система збалансована за навантаженням, з її зростанням не настає деградація продуктивності системи (швидкість генерації сторінок сервером практично не змінюється, це видно на червоному графіку);

– збільшується час віддачі сторінок клієнтам, зростає черга (синій графік).

Другий тест – емуляція аварії (відключення) сервера з базою даних MySQL (рис. 3.23).

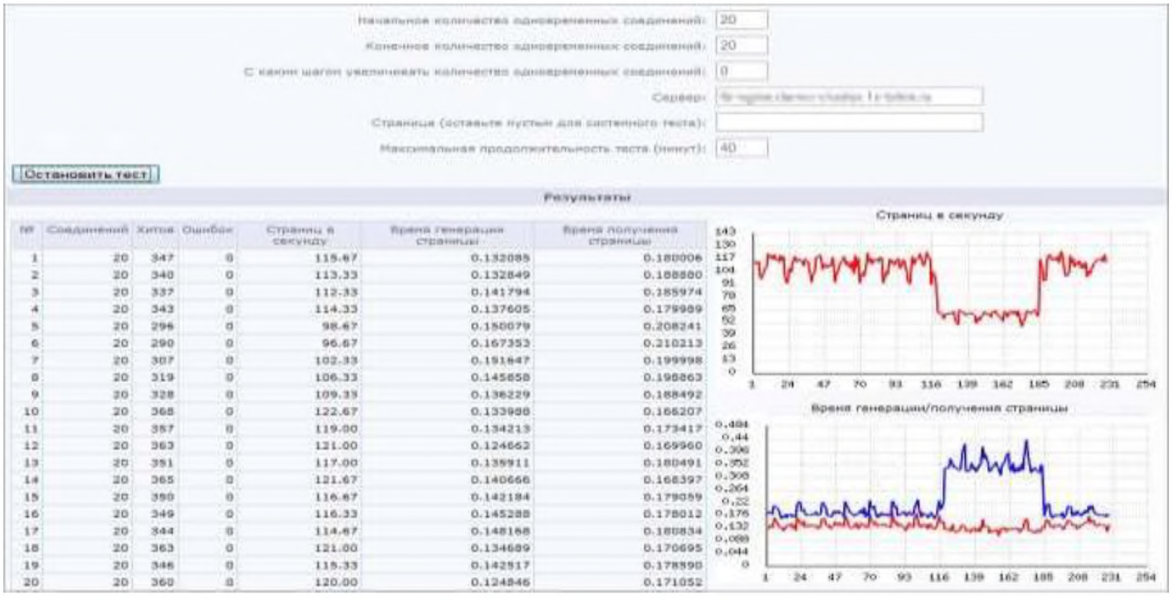


Рисунок 3.23 – Тест – емуляція відключення сервера з базою даних MySQL

У другому тесті емулюємо відключення однієї з машин кластера (та, на якій працює slave-база даних MySQL). Невеликі піки на графіках – моменти синхронізації контенту (у нас у тесті встановлений період оновлень, його можна зменшити). Великий «провал» («сплеск» на синьому графіку) – момент відключення однієї з машин кластера.

Третій тест – емуляція аварії (вимкнення) сервера з master базою даних MySQL (рис. 3.24).

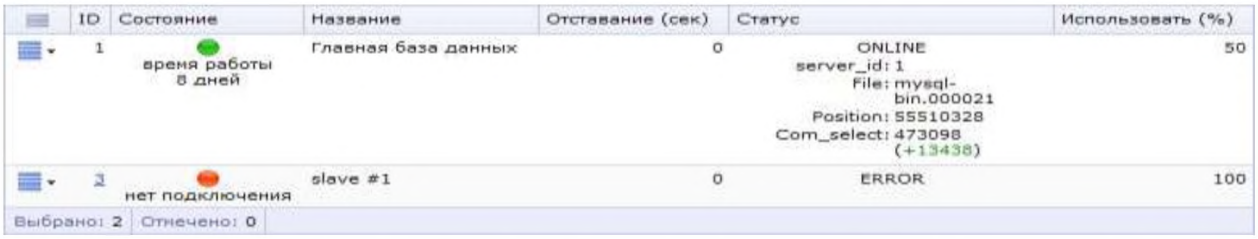


Рисунок 3.24 – Тест – емуляція аварії сервера з master базою даних MySQL

Вимкнення сервера з master-базою даних MySQL відрізняється від попереднього тесту тим, що необхідно виконати ряд дій для перемикання бази даних slave в режим майстра.

3.10 Варіанти конфігурації вебкластера для вирішення практичних завдань

Насамперед, необхідно уважно проаналізувати (спрогнозувати) характер навантаження на вебдодаток. Для збору та аналізу даних рекомендується проактивно використовувати відомі інструменти моніторингу, такі як munin, zabbix, apache server-status тощо.

Нижче наведено варіанти конфігурації вебкластера для різних типів навантаження.

1. Високе навантаження на процесори, невисоке/середнє навантаження на СУБД, контент змінюється рідко. Необхідно знизити навантаження на процесори –

підключаємо до вебкластер ноду(и). Ноди заводимо під балансувальник навантаження. В результаті навантаження на процесори розпаралелюється між нодами. Рекомендується також на кожній ноді запустити сервер memcached – це дозволить ще більше знизити навантаження на процесори за рахунок того, що кеш створюватиметься один раз на одній із нод вебкластера, а використовуватиметься решта нодів. Для синхронізації контенту підійде csync2, nfs/cifs-сервер на одній із нод вебкластера.

2. Високе зростаюче навантаження на СУБД. Для розвантаження СУБД необхідно організувати master-slave реплікацію. Чим більше буде додано slave-нод, тим сильніше знизиться навантаження на основну master базу даних. Якщо slave-ноди різної потужності, рекомендується адекватно розподілити між ними навантаження за допомогою параметра Відсоток розподілу навантаження (0..100). В результаті навантаження на СУБД розподілиться між master та slave нодами вебкластера. При подальшому збільшенні навантаження на СУБД рекомендується:

– оптимізувати master і slave-ноди. Т.к. master-сервер буде використовуватися переважно для запису, його потрібно шляхом налаштування параметрів оптимізувати для запису, а slave-ноди – оптимізувати для читання.

– додавати slave-ноди.

3. Великий обсяг кешу, кеш часто перебудовується, контент змінюється часто, змін багато. Потрібно налаштувати ефективну роботу з кешем. Для цього запускаємо на кожній ноді сервер memcached, виділяємо кожному по кілька GB пам'яті (залежно від вимог програми). Для синхронізації контенту рекомендується використовувати виділений nfs/cifs-сервер або ocfs/gfs або аналоги.

Аналіз характеру навантаження та вибір оптимальної конфігурації вебкластера є критичними для ефективності системи при різних типах завдань та вимог. Коректне врахування цих аспектів дозволяє досягти оптимальної роботи кластера та забезпечити його стабільність у різних умовах навантаження. Урахування особливостей навантаження на процесори, СУБД та великого обсягу кешу дозволяє вибрати оптимальні параметри конфігурації кластера, забезпечуючи оптимальний розподіл ресурсів та ефективне використання його складових частин

для досягнення бажаного результату. Необхідно пам'ятати, що налаштування конфігурації вебкластера має бути адаптоване до конкретних вимог та потреб системи, забезпечуючи не лише оптимальну роботу в поточний момент, але й гнучкість для майбутніх змін та розвитку.

3.11 Вибір та обґрунтування методики оцінки економічної ефективності

Аналіз основних показників порівняльної ефективності створення та функціонування інформаційних систем включає визначення наступних показників.

1. Річна економія поточних витрат, отримана від функціонування системи.
2. Додаткові капітальні вкладення (КВ), необхідні для створення системи.
3. Термін окупності додаткових КВ.
4. Розрахунковий коефіцієнт ефективності додаткових КВ.
5. Річний економічний ефект.
6. Річна економія трудових витрат на обробку даних у системі.

Оцінимо ці показники.

1. Річна економія поточних витрат складається з прямої економії та інших складових. Пряма економія виникає від автоматизації обробки інформації.

$$\Delta Ст = \Delta Сп + \Delta Сн, \quad (3.1)$$

де $\Delta Сп$ – пряма економія,

$\Delta Сн$ – непряма економія (у цій роботі не розраховується).

Пряма економія, яку отримують від автоматизації обробки інформації розраховується так:

$$\Delta Сп = \Delta Сб - \Delta Спор, \quad (3.2)$$

де $\Delta Сб$ – базовий період, який береться до застосування системи,

$\Delta Спор$ – порівнюваний період, коли система працює у режимі автоматизації.

Розрахунок показника за порівнюваний період містить:

$$\Delta Спор = С1 + С2 + С3 + С4 + С5 + Спр, \quad (3.3)$$

де С1 – витрати на оплату праці персоналу;

С2 – нарахування на фонд оплати праці (податок 22%);

С3 – витрати сировину, матеріали (картриджі, папір, комплектуючі);

С4 – амортизація обладнання, зазвичай розглядається як лінійна з терміном служби від 3 до 8 років;

С5 – інші витрати (витрати на відрядження, інформаційні витрати, плата за кредит, податки, представницькі витрати);

Спр – передвиробничі витрати, які потрібно визначити додатково.

Передвиробничі витрати – витрати, які можуть бути направлені на розробку (купівлю) програмних засобів, на навчання фахівців і т.д.

2. Додаткові капітальні вкладення охоплюють будівництво, оренду приміщення, ремонт, придбання мережевого обладнання тощо.

3. Термін окупності капітальних вкладень розраховується за формулою:

$$Т = КД / \Delta Ст, \quad (3.4)$$

де ΔСт – річна економія поточних витрат,

КД – капітальні вкладення, наведені до 1 року

4. Розрахунковий коефіцієнт ефективності Ер є величина зворотна Т, Ер = 1/Т. Ен = 0,33 – нормативний коефіцієнт ефективності.

Якщо розрахунковий коефіцієнт більший чи дорівнює Ен, то проект приймається до впровадження, тобто створення АІС ефективно.

5. Річний економічний ефект розраховується за формулою:

$$Е = \Delta Ст – КД * Ен, Ен = 0,15 \quad (3.5)$$

6.Річна економія трудових витрат розраховується як різниця між базовим та порівняльним періодами до та після впровадження системи в режимі автоматизації:

$$\Delta Т = \Delta Тб – \Delta Тпор, \quad (3.6)$$

де ΔТб – період базовий до застосування системи,

ΔТпор – період, порівнюваний, тобто період роботи системи у режимі автоматизації.

Далі було виконано розрахунок основних економічних показників від впровадження інформаційної системи. Результати розрахунків представлені в табл. 3.3.

Таблиця 3.3. Результати розрахунку базових економічних показників

Затрати	Базовий період (ΔСб), грн.	Порівнюваний період (ΔСпор), грн.
С1	520000	400000
С2	244900	203500
С3	2250	3500
С4	0	2250
С5	500	500
Спр		2000
Всього	767650	611750

ΔСп = 767650 – 611750 = 155900 грн.

ΔСт = 155900 грн.

2. Додаткові капітальні вкладення.

Оренда приміщення (на рік) = 4000 грн · 60 м^2 = 240000 грн.

Придбання комп'ютера (Комплектація: AMD Phenom II X6 1075, 8Гб, 2000Гб, GeForce GTX 560) = 13000 грн.

Придбання монітора (17» Acer V173Ab чорний 5ms) = 3500 грн.

Придбання принтера Canon LBP-7800 = 3800 грн.

Термін функціонування системи = 10 років.

Сума додаткових капітальних вкладень за рік

$$240000 + (13000 + 3500 + 3800) / 10 = 242030 \text{ грн.}$$

3. Термін окупності капітальних вкладень:

$$T = 242030 / 155900 = 1,55$$

4. Розрахунковий коефіцієнт ефективності Ер.

$$Ep = 1/1,55 = 0,64$$

Проект ефективний.

5. Річний економічний ефект

$$E = 155900 – 242030 \cdot 0,15 = 119555 \text{ грн.}$$

6. Річна економія трудових витрат:

$$\Delta T = 520000 – 240000 = 280000 \text{ грн.}$$

Розрахунок показав, що застосування системи принесе підприємству вигоду, ефект від використання системи становитиме 119555 грн на рік.

Висновки до розділу 3

У третьому розділі на прикладі хмарного вебкластеру було розглянуто процес налаштування та тестування віртуалізованої розподіленої обчислювальної системи. Розглянуті кроки створення і налаштування віртуальних машин для вебкластера, встановлення рекомендованих параметрів безпеки для відповідних сервісів, таких як memcached і csync2. Детально описано кожен крок створення віртуальної машини, вибір необхідної апаратної конфігурації, імпорт ключів шифрування для безпечного доступу, а також створення груп безпеки для налаштування прав доступу. Окрім цього, вказано на необхідність закриття доступу ззовні до певних портів для забезпечення безпеки.

Також було розглянуто налаштування переключення між серверами при реплікації даних, яке передбачає кілька кроків, включаючи припинення потоку отримання оновлень бінарного лога на slave-серверах, підготовку нового master-сервера та переключення slave-сервера на новий master-сервер. Крім того, важливо правильно налаштувати вебдодаток для використання нового master-сервера та відновити доступ клієнтів до вебпрограми.

Виконано аналіз основних показників порівняльної ефективності створення та функціонування інформаційних систем. Розрахунок показав, що застосування запропонованої розподіленої системи принесе підприємству вигоду, ефект від використання системи становитиме 119555 грн на рік.

ВИСНОВКИ

У роботі була поставлена і вирішена актуальна наукова задача розробки та тестування програмного комплексу на основі віртуалізованої системи для вирішення різних обчислювальних задач з використанням принципів розподілених обчислень.

У ході виконання кваліфікаційної роботи було проведено дослідження стану розвитку галузі розподілених та хмарних обчисленб. На основі роботи можна сформулювати наступні висновки.

1. Визначення розподілених систем має різноманітні формулювання, але жодне з них не є строгим чи загальноприйнятим. Одне з визначень розподіленої системи – це система, в якій взаємодія та синхронізація програмних модулів відбуваються через передачу повідомлень між незалежними комп'ютерами мережі. Розподілені системи відрізняються від централізованих своєю асинхронністю в роботі вузлів, відсутністю спільної пам'яті та географічним розподілом, а також гетерогенністю та незалежністю компонентів, що можуть виконувати загальне завдання взаємопов'язано, надаючи послуги одне одному.

2. Проаналізовано основне завдання, яке ставиться перед розподіленими системами – полегшити користувачам доступ до віддалених ресурсів та забезпечити їхнє спільне використання, регулюючи цей процес. Розглянуто дві основні класифікації розподілених систем: за розмірами та способами управління, а також за функціональними характеристиками (типом ресурсів, видами завдань та оптимізацією). Використовуючи подвійну класифікацію можна чіткіше окреслити межі використання та застосовності системи.

3. Системи розподілених обчислень можуть використовувати різні підходи, такі як кластери високої доступності з різними принципами відмовостійкості, мережеве розподілення навантаження та серверні ферми, щоб забезпечити стабільну та ефективну роботу у високонавантажених умовах. Архітектура кластерів, таких як Beowulf, та системи HPC використовуються для досягнення високої продуктивності, тоді як грід-системи розбивають завдання на незалежні

процеси, забезпечуючи ефективне використання ресурсів.

4. Хмарні обчислення представляють собою перспективну технологію, що стала не тільки невід'ємною частиною повсякденного життя користувачів Інтернету, але й гнучким та потужним інструментом для бізнесу, який значно спрощує доступ до обчислювальних ресурсів та забезпечує ефективне їх використання, надаючи широкий спектр можливостей.

5. Визначено два основних типи мережевих операційних систем: однорангові та дворангові, які відрізняються у розподілі функцій між комп'ютерами у мережі. Мережеві операційні системи мають простішу структуру порівняно з розподіленими, проте вони не досягають повної прозорості, яка характерна для розподілених ОС. Недоліками мережевих операційних систем є складність у використанні та управлінні, хоча вони надають можливість легко додавати, замінювати чи видаляти нові комп'ютери завдяки їхній незалежності у мережі.

6. Розглянуті кроки створення і налаштування віртуальних машин для вебкластера, встановлення рекомендованих параметрів безпеки для відповідних сервісів, таких як memcached і csync2. Детально описано кожен крок створення віртуальної машини, вибір необхідної апаратної конфігурації, імпорт ключів шифрування для безпечного доступу, а також створення груп безпеки для налаштування прав доступу. Окрім цього, вказано на необхідність закриття доступу ззовні до певних портів для забезпечення безпеки.

Запропоновано варіанти конфігурації вебкластера для вирішення практичних завдань та різних типів навантаження, а саме: а) високе навантаження на процесори, невисоке/середнє навантаження на СУБД, контент змінюється рідко; б) високе зростаюче навантаження на СУБД; в) великий обсяг кешу, кеш часто перебудовується, контент змінюється часто. Аналіз характеру навантаження та вибір оптимальної конфігурації вебкластера є критичними для ефективності системи при різних типах завдань та вимог.

Таким чином, поставлені задачі розв‘язано у повному обсязі. Напрямком подальших досліджень є розробка аналітичних моделей функціонування хмарної архітектури з врахуванням атак на доступність та цілісність даних в системі.