

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра _____ комп'ютеризованих систем управління _____

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.

«____» _____ 2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: _____ Онлайновий репозиторій проєктів компанії розробника
_____ програмного забезпечення _____

Виконавець: _____ Орнатська Є.Д. _____

Керівник: _____ Росінська Г.П. _____

Нормоконтролер: _____ Тупота Є.В. _____

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 "Комп'ютерна інженерія"
(шифр, найменування)

Освітньо професійна програма «Системне програмування»
Форма навчання заочна

ЗАТВЕРДЖУЮ

Завідувач кафедри

Литвиненко О. Є.

« » 2021 р.

ЗАВДАННЯ на виконання дипломного проєкту

Орнатської Євгенії Денисівни
(прізвище, ім'я, по батькові)

1. Тема роботи: “Онлайновий репозиторій проєктів компанії розробника програмного забезпечення”

затверджена наказом ректора від "21" грудня 2020 року № 2523 /ст.

2. Термін виконання роботи: з 11.01.2021 до 28.02.2021

3. Вихідні дані до проєкту (роботи): постановка задачі до виконання роботи, мови програмування: C++, СУБД: MySQL.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1) принципи оцінювання якості кода в програмній інженерії;

2) методи оцінки якості програмного коду;

3) розробка програмного забезпечення для оцінки якості коду.

5. Перелік обов'язкового графічного матеріалу:

1) діаграма змін у програмній системі, орієнтованій на кінцевих користувачів;

2) візуалізація принципу роботи системи контролю версій;

3) діаграма взаємодії модуля «Репозиторій» в системі контролю версій;

4) вікно результатів команди порівняння версій;

5) схема алгоритму обробки запиту на пошук у репозиторії.

6. Календарний план-графік

№ п/п	Етапи виконання дипломного проєкту	Термін виконання етапів	Примітка
1	Провести аналіз літератури за темою дипломного проєкту та аналіз існуючих систем	11.01.21 13.01.21	
2	Зробити вибір компонентів системи	14.01.21- 15.01.21	
3	Розробити структуру програмних засобів системи	16.01.21- 17.01.21	
4	Розробити програмні засоби	18.01.21- 27.01.21	
5	Провести відладку програмних засобів на модельному зразку	28.01.21- 02.02.21	
6	Написати пояснювальну записку	03.02.21- 13.02.21	
7	Підготувати презентацію	14.02.21- 16.02.21	
8	Оформити супроводжувальну документацію	17.02.21 19.02.21	

7. Дата видачі завдання « 11 » січня 2021 р.

Керівник дипломного проєкту _____ Росінська Г.П.
(підпис)

Завдання прийняв до виконання _____ Орнатська Є.Д.
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту “Онлайновий репозиторій проектів компанії розробника програмного забезпечення”: 72 с., 34 рис., 26 літературних джерела, 1 додаток.

WEB-ПРОЕКТ, WEB-РЕСУРС, РЕПОЗИТОРІЙ, ПРОГРАМНИЙ ПРОЕКТ, ОНОВЛЕННЯ КОДУ, ПРОГРАМНИЙ КОД

Мета дослідження – розробити онлайновий репозиторій проектів компанії розробника програмного забезпечення з підтримкою системи контролю версій програмного забезпечення на віддалених серверах.

Об’єкт дослідження – оновлення програмного забезпечення.

Предмет дослідження – онлайновий репозиторій проектів компанії розробника програмного забезпечення.

Встановлено, що запропонований метод оновлення версій програмного забезпечення *web*-ресурсів, який використовується в репозиторії ПЗ, побудовано за парадигмою сучасних *SCV* і має сенс продовжувати роботу в даному напрямку.

Результати дипломного проекту рекомендується використовувати при командній розробці програмних модулів веб-проектів. Розвиток проекту передбачається за рахунок розширення параметрів оновлення ПЗ.

3MICT

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

RDBMS	Реляційна система управління базами даних.
TPC-тести	Базовий показник ефективності обробки транзакцій - визначте ефективність транзакцій системи баз даних.
API	Інтерфейс прикладних програм.
AWS	Amazon Web
БАЗА	Базова доступність, м'який стан, евентуальна узгодженість
Тест TPC-W	Бенчмарк транзакційної веб-електронної комерції, запроваджений Радою з питань обробки транзакцій. TPC-W визначає робоче навантаження електронної комерції, що імітує діяльність роздрібного веб-сайту, що створює велике навантаження на серверну базу даних.
WIPS	Взаємодія в Інтернеті за секунди

ВСТУП

Актуальність.

Протягом останніх десяти років новий середній Інтернет залучив дуже багато людей, в результаті чого веб-додатки використовуються десятками тисяч користувачів і навіть мільйонами, якщо додаток виявиться успішним. З такою кількістю користувачів обсяг даних, які потрібно зберігати, і кількість запитів даних, на які потрібно обробляти, величезний і може швидко стати обмежувальним фактором. Традиційні реляційні системи баз даних та старіші паралельні та розподілені системи баз даних часто можуть недостатньо масштабуватися до таких складних сценаріїв. Традиційні системи баз даних мають намір забезпечити узгодженість у будь-який час, і коли вони розподіляються, намагаються досягти прозорості розподілу. Це означає, що вони спрямовані на те, щоб створити у кінцевого споживача враження, що існує лише одна система замість низки спільних систем. Вони навіть доходять до того, що скоріше зазнають невдачі, ніж порушують цю прозорість. Це призводить до того, що традиційна система баз даних недоступна, коли один з її розділів або підсистем недоступний.

З точки зору постачальників дуже затребуваних веб-сервісів, найважливішою властивістю, яку вони просять у системі зберігання, є доступність, і вони не можуть терпіти простій у їх обслуговуванні. Це призводить до цікавої тенденції того, що постачальники надзвичайно успішних послуг впроваджують власні рішення для хмарного сховища та відмовляються від традиційних СУБД для цих додатків. Прикладами є Amazon S3, Yahoo PNUTS, Google BigTable та багато інших. Ці системи зберігання призначені для надійного зберігання величезних обсягів даних та масштабування до тисяч машин, щоб бути постійно доступними.

Як показує відома теорема CAP, є компроміси. Ви можете мати не більше двох властивостей - узгодженість, доступність і толерантність до мережевих архівів одночасно в будь-якій спільній системі даних. Згадані хмарні рішення

для зберігання скорочують консистенцію, часто лише підтримуючи можливу послідовність для збереження двох інших властивостей. Але ця розслаблена узгодженість не завжди бажана або всілякі сценарії використання, тоді як інші з нею добре.

Ще одним недоліком поточних хмарних служб зберігання даних є інтерфейс, який вони надають для вставки та доступу до даних з точки зору користувача. Вони часто надають лише дуже простий інтерфейс, де можна отримати доступ та вставити лише неструктуровані значення, що зберігаються під певним ключем або подібними основними методами. Якщо кінцеві користувачі хочуть отримати більш складні та узагальнені результати, вони повинні збирати окремі фрагменти даних по системах зберігання та поєднувати їх самі. Це великий крок назад від традиційних СУБД, які часто підтримують розширену мову запитів та маніпуляцій як інтерфейс до своїх даних, забезпечують використання схеми та намагаються мінімізувати дублювання даних за допомогою передової практики та звичайних форм.

Це підводить нас до тем даної магістерської роботи, яка намагається об'єднати ці два підходи до рішень для зберігання даних і має намір реалізувати SQL-інтерфейс поверх хмарної системи зберігання. Більш конкретно я прагну використовувати Amazon S3 як рівень зберігання традиційної бази даних. Завдяки еластичності системи S3, яка може додавати та видаляти вузли зі свого запущеного кластера, ми отримали б високо масштабовану систему баз даних, де масштабованість походить від рівня зберігання. Використовуючи традиційну базу даних як основу, ми все ще можемо зберегти її рівень SQL і повторно використовувати мову запитів та скористатися багатьма внутрішніми алгоритмами.

До нашої нової системи баз даних можна отримати доступ за допомогою стандартних API та бібліотек підключень бази даних. Це дозволяє багатьом існуючим стороннім програмам використовувати нашу систему без змін. Крім того, вже існуюче програмне забезпечення можна зробити масштабованим, перемикаючи базовий рівень зберігання бази даних.

РОЗДІЛ 1

ХМАРНІ ОБЧИСЛЕННЯ ДЛЯ ОНЛАЙНОВОГО РЕПОЗИТОРІЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ПІДТРИМКОЮ СИСТЕМИ КОНТРОЛЮ ВЕРСІЙ

1.1 Що таке хмарні обчислення?

Коріння хмарних обчислень - у корисних та мережевих обчисленнях [17]. Обчислювальні програми вивчаються з 90-х років, наприклад, за проектом OceanStore в UC Berkeley. Однак, мабуть, він мав найбільший успіх як мережеві обчислення в науковому співтоваристві [14]. Сіткові обчислення були розроблені для цілком конкретних цілей - здебільшого, для запуску великої кількості процесів аналізу наукових даних. Amazon представив цю ідею в маси і сильно вплинув на термін хмарні обчислення.

Хмарні обчислення характеризуються зовнішнім доступом до спільних ресурсів за замовленням. Це стосується як послуги, що надається через Інтернет, так і апаратного та програмного забезпечення в центрах обробки даних, які надають ці послуги. Хоча існує не так багато згоди щодо точного визначення хмарних обчислень [1, 17, 28, 29], більшість погоджуються щодо наступних чотирьох аспектів, які в цій формі є новими для хмарних обчислень:

На вимогу: вимагання ресурсів на вимогу, тим самим усуваючи необхідність попередніх інвестицій та плануючи заздалегідь забезпечення обладнання.

Оплата за користування: пропозиція оплатити використані обчислювальні ресурси комунальним способом (наприклад, процесорні одиниці на годину, зберігання на день), подібно до рахунку за електроенергію.

Кафедра КСУ				НАУ 21 08 21 000 ПЗ			
Виконав	Орнатська С.Д.			Призначення та організаційна структура онлайн-репозиторію програмного забезпечення з підтримкою системи контролю версій	Літера	Аркуш	Аркушів
Керівник	Росінська Г.П.				Д	9	72
Консульт.					СП 501Бз 123		
Норм. контр.	Тупота С.В.						
Зав. Каф.	Литвиненко О.Є.						

Масштабованість: ілюзія наявності нескінченних обчислювальних ресурсів на вимогу, що позбавляє потреби хмарних обчислень планувати заздалегідь.

Технічне обслуговування та стійкість до несправностей: Якщо компонент виходить з ладу, за заміну компонента несе відповідальність постачальник послуг. Крім того, системи побудовані надзвичайно надійно, часто тиражуються в декількох центрах обробки даних, щоб мінімізувати відключення.

Хоча іноді його розглядають як ажіотаж, головним успіхом хмарних обчислень є не технологічний, а економічний. Хмарні обчислення дозволяють компаніям передавати ІТ-інфраструктуру на аутсорсинг і, таким чином, отримувати прибуток від економії на масштабі та ефекту впливу аутсорсингу [1]. Крім того, хмарні обчислення перекладають ризик забезпечення на хмарних провайдерів, що ще більше зменшує вартість. З цієї причини навряд чи це лише короткострокова тенденція.

Економічні аспекти також пояснюють нову тенденцію до побудови приватних хмар. На відміну від загальнодоступних хмар, які зазвичай називають загальним терміном хмарних обчислень, які пропонують загальнодоступну послугу, приватні хмари означають менші установки, розміщені всередині компанії, що пропонує внутрішні послуги. У цьому контексті використовується термін хмара, оскільки хмарні обчислення створили нове мислення щодо розробки програмного забезпечення. Тобто основна увага приділяється спочатку низькій вартості, масштабованості, відмовостійкості та простоті обслуговування, а лише потім продуктивності, часу відгуку, функціональності тощо. Таким чином, програмне забезпечення, розроблене в контексті хмарних обчислень, може також допомогти зменшити вартість локальних розгортань. Це особливо бажано, якщо проблеми довіри або безпеки перешкоджають використанню загальнодоступних хмар.

Постановка проблеми

Хмарні обчислення - все ще досить нова сфера, яка ще не повністю визначена. В результаті існує багато цікавих дослідницьких проблем, які часто

поєднують різні галузі досліджень, такі як бази даних, розподілені системи чи операційні системи. Ця дисертація зосереджена на тому, як створювати веб-додатки для баз даних поверх хмарної інфраструктури. Зокрема, такі проблеми розглядаються вище.

Інфраструктура прикладних програмних інтерфейсів (API): Сьогодні пропозиції хмарних послуг значно різняться, і жодного стандарту не існує. Таким чином, портативність між службами не гарантується. Навіть не існує єдиної думки, які саме послуги є. Визначення мінімального набору сервісів та відповідного API, який би дозволяв створювати вдосконалені додатки, значно зменшує тягар переміщення додатків у хмару.

Такий довідковий API важливий як основа для всіх подальших розробок у цій галузі.

Архітектура: Хоча все більше веб-додатків переміщуються до хмари, правильна архітектура все ще не визначена. Оптимальна архітектура дозволить зберегти масштабованість та доступність комунальних послуг та досягти того ж рівня узгодженості, що і традиційні системи баз даних (тобто транзакції ACID). На жаль, неможливо мати все це, як стверджує теорема про пивоварну CAP (узгодженість, доступність, толерантність до розділів). Теорія CAP доводить, що доступність, толерантність до мережевих розділів та послідовність неможливо досягти одночасно [19]. З огляду на той факт, що у більших системних мережевих розділах трапляються, більшість служб інфраструктури жертвують послідовністю на користь доступності [6, 9]. Якщо програма вимагає більшої узгодженості, її потрібно впровадити зверху.

Послідовність застосування: Послідовність відіграє важливу роль у контексті хмарних обчислень. Це не тільки безпосередньо впливає на доступність, але також впливає на продуктивність та вартість. У великомасштабних системах висока узгодженість передбачає більше повідомлень, які доводиться надсилати між серверами, якщо ідеальне розділення неможливе. Тут ідеальне розділення відноситься до схеми розділення, де всі можливі транзакції можуть бути виконані на одному сервері системи. У більшості додатків (наприклад, веб-магазин, програми соціальних мереж,

бронювання авіарейсів тощо) це неможливо, якщо слід зберегти масштабованість за допомогою масштабування. Наприклад, у веб-магазині всі товари можуть бути в одній кошику для покупок. Таким чином, розділити веб-магазин за продуктами неможливо, але інші розділи (наприклад, замовник) також не допомагають, як знову ж таки,

Мови програмування: Ще однією проблемою переміщення додатків у хмару є необхідність оволодіти кількома мовами [21]. Багато програм покладаються на серверну систему, що працює під керуванням SQL або інших (спрощених) мов. На стороні клієнта використовується JavaScript, вбудований в HTML, і сервер додатків, що стоїть між серверною базою та клієнтом, реалізує логіку за допомогою якоїсь мови сценаріїв (наприклад, PHP, Java та Python). Усі рівні зазвичай спілкуються за допомогою XML-повідомлень. Для подолання різноманітності мов XQuery / Xscript вже пропонується як уніфікована мова програмування, яка може працювати на всіх рівнях [5, 7, 16].

1.1.1 Огляд хмарних служб

Хмарні служби можна поділити на три типи: Інфраструктура як послуга (IaaS), платформа як послуга (PaaS) та програмне забезпечення як послуга (SaaS). Це також показано на малюнку 1.1.

Інфраструктура як послуга (IaaS)

IaaS - найзагальніша форма хмарних сервісів. Це стосується служб нижчого рівня, таких як доступ до віртуальних машин, служб зберігання даних, баз даних або сервісів черги, які необхідні для створення середовища додатків з нуля. IaaS полегшує це

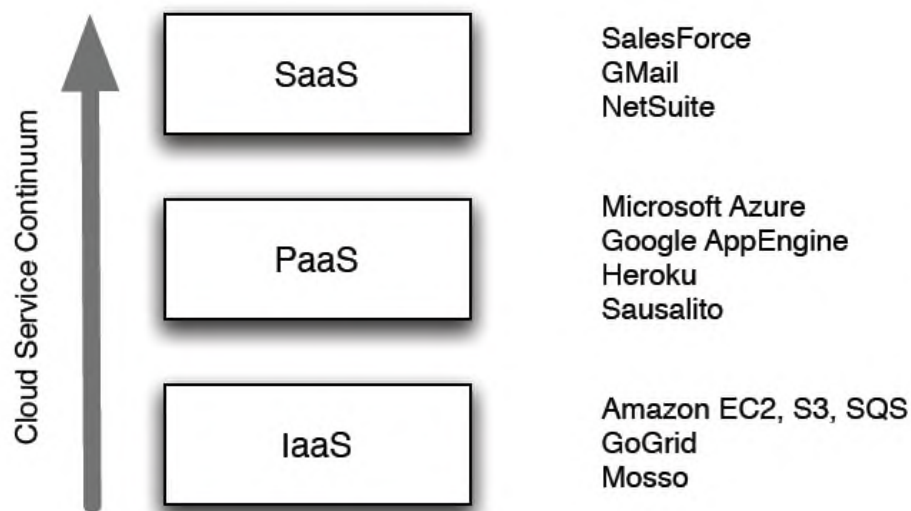


Рисунок 1.1: Хмарний континуум

забезпечити та дозволити собі такі ресурси, як сервери, мережа та сховище. Послуги оплачуються, як правило, за кількістю спожитих ресурсів. IaaS забезпечує найбільшу свободу в розробці додатків, одночасно вимагаючи від користувачів роботи з деталями нижчого рівня, такими як віртуальні машини, операційні системи, виправлення тощо. Трьома найбільшими провайдерами IaaS сьогодні є Amazon, GoGrid та Mosso. Послуги, пропоновані цими постачальниками, наведені в таблиці 1.1.

Послуги хостингу: Найбільш універсальними послугами інфраструктури, які пропонують провайдери, є послуги хостингу, які дозволяють клієнтам орендувати машини (CPU + диски) на певний проміжок часу. Прикладами таких служб є Amazon EC2, хмарний хостинг GoGrid та хмарний сервер Rackspace. Технічно клієнт отримує віртуальну машину (VM), яка розміщена на одному із серверів провайдера. Ціни на послуги, як правило, складають від 0,01 до 1 доларів США на годину та, наприклад, залежно від конфігурації (наприклад, пам'ять, потужність центрального процесора, дисковий простір), а також додаткові витрати на мережевий трафік, які зазвичай коштують від 0,1 до 0,5 доларів США за ГБ. Віртуальні машини здатні розміщувати майже всі види Linux та Windows, а отже, дозволяють встановлювати та експлуатувати майже всі види програмного забезпечення в хмарі. Найбільшою перевагою хостинг-послуг є

можливість збільшення та зменшення кількості віртуальних машин на вимогу. Наприклад, якщо програма вимагає більше ресурсів протягом обіду, ці ресурси можна додати у вигляді додаткових віртуальних машин. Хоча більшість провайдерів допомагають розподілити навантаження по серверах за допомогою базових механізмів балансування навантаження, програмісту все одно потрібно подбати про сесии, збереження даних, узгодженість, кеш-пам'яті тощо. Крім того, служба

Таблиця 1.1. Інфраструктура як постачальники послуг та продукти

П остачал ьник	Послуга хостингу	Служба зберігання	Інші послуги
А мазонка	Elastic Compute Cloud (EC2)	<ul style="list-style-type: none"> • Магазин еластичних блоків (EBS) • Проста послуга Storage (S3) 	<ul style="list-style-type: none"> • Просте обслуговування черг (SQS) • CloudFront • SimpleDB
Se rverPath	Хмарний хостинг GoGrid	GoGrid Cloud Storage	-
С тійка	Хмарні сервери	Хмарні файли	-

постачальники не гарантують високу надійність для віртуальних машин. Тобто, якщо віртуальна машина аварійно завершує роботу (або спричинена користувачем або постачальником), втрачається повний стан віртуальної машини, включаючи стан локального приєднаного диска.

Послуги зберігання: Щоб зберегти дані протягом усього життя віртуальної машини, постачальники послуг пропонують окремі послуги зберігання, такі як Amazon Simple Storage Service або Elastic Block Store, GoGrid Cloud Storage та Mosso's Cloud Files. Знову ж таки, ці послуги коштують утилітно, сьогодні близько 0,1 - 0,3 дол. США за ГБ на місяць, плюс вхідний та вихідний мережевий трафік і досить часто додаткові платежі за запит на читання та читання. Самі послуги відрізняються між собою ступенем паралельності, який вони дозволяють (один користувач на об'єкт одночасно проти кількох користувачів на один об'єкт), гарантіями узгодженості (сильні гарантії ACID проти можливої послідовності), ступенем надійності (єдиний центр обробки даних проти декількох реплікація центру обробки даних) та функції (прості ключ-значення проти запитів).

Інші послуги: Поряд із основними послугами (тобто хостингом та зберіганням), Amazon пропонує додаткові послуги, щоб допомогти розробникам у створенні своїх додатків всередині хмари. Найголовніше, що SimpleDB пропонує простий спосіб зберігання та отримання структурованих даних. Однак назва вводять в оману, оскільки SimpleDB є скоріше індексом, ніж базою даних: поняття транзакції не існує, єдиним типом даних є текст, а запити обмежуються простими предикатами, включаючи сортування та підрахунок, але виключаючи об'єднання або більш складну функціональність групування. Детальніше SimpleDB описано в розділі 2.2.5. Поряд з SimpleDB Amazon пропонує послугу простої черги (SQS). Знову ж таки, назва вводять в оману, оскільки SQS не пропонує черг із семантикою «перший у першому». Натомість, SQS - це послуга обміну повідомленнями рівно одного разу, де повідомлення можна помістити в так звану "чергу" і отримати з неї. Розділ 2.2.4 розглядає SQS більш докладно. Останньою послугою в групі інфраструктурних пропозицій Amazon є CloudFront, служба доставки контенту. Це дозволяє розробникам ефективно доставляти вміст за допомогою глобальної мережі крайових серверів. CloudFront спрямовує запит на об'єкт до найближчого краю і, таким чином, зменшує затримку.

Незважаючи на те, що можливо поєднувати послуги різних провайдерів, наприклад, Amazon S3, із послугою хостингу GoGrid, такі комбінації є рідкістю,

оскільки вартість мережевого трафіку та затримки роблять їх дорожчими та повільнішими. Крім того, поки що між хмарними провайдерами не існує стандартного API, що ускладнює міграцію між постачальниками послуг.

Платформа як послуга (PaaS)

Пропозиції PaaS забезпечують платформу розробки більш високого рівня для написання додатків, приховуючи від розробника деталі низького рівня, такі як операційна система або балансування навантаження. Платформи PaaS часто будуються поверх IaaS і обмежують розробника єдиною (обмеженою) мовою програмування та заздалегідь визначеним набором бібліотек - так званою платформою. Найбільша перевага PaaS полягає в тому, що розробник може повністю сконцентруватися на бізнес-логіці, не маючи справу з деталями нижчого рівня, такими як виправлення для операційної системи або конфігурації брандмауера. Крім того, масштабованість забезпечується автоматично, якщо розробник дотримується певних вказівок. Ці обмеження становлять основний мінус PaaS. Це особливо вірно, якщо частини програми вже існують або потрібні певні бібліотеки. Знову ж таки, не існує стандартів між провайдерами PaaS. Отже, зміна постачальника послуг є ще складнішою та дорожчою, ніж із IaaS, оскільки код програми сильно залежить від мови хосту та API.

Найвизначнішими хмарними платформами є App Engine від Google, Azure Platform від Microsoft та Force.com (див. Таблицю 1.2). Google Engine Engine пропонує розміщувати мови на Java та Python, тоді як платформа Microsoft Azure підтримує платформу .NET та PHP. Як Microsoft, так і Google обмежують код, який може працювати на їхній службі. Наприклад, Google Engine Engine не дозволяє програмам Java створювати потоки, записувати дані в локальну файлову систему, встановлювати довільні мережеві з'єднання або використовувати прив'язки JNI.

Подібно до IaaS, збереження даних знову є критичним, і обидві платформи пропонують для цього спеціальні послуги. DataStore від Google дозволяє зберігати структуровані та напівструктуровані дані, підтримує транзакції та навіть має просту мову запитів, яка називається GQL. Тим не менше, DataStore не надає тієї ж функціональності, що і "повна" база даних. Наприклад, підтримка

транзакцій є досить елементарною, і GQL, хоча як обмежена мова, подібний до SQL, не має функцій для об'єднань та складних агрегатів. З іншого боку, база даних SQL Azure - це невелика база даних MS SQL Server, яка надає функції MS SQL Server. Однак його розмір обмежений (на даний момент максимум 10 ГБ).

Таблиця 1.2. Платформа як постачальники послуг та продукти

Платформа	Середовище виконання	Служба зберігання	Додаткові послуги
Google App Engine	<ul style="list-style-type: none"> • Java • Python 	Магазин даних	<ul style="list-style-type: none"> • Облікові записи Google • Маніпулювання зображеннями • Пошта • Memcache
Платформа Microsoft Azure	Windows Azure: .Net мова	<ul style="list-style-type: none"> • База даних SQL Azure • Служба Azure Storage 	<ul style="list-style-type: none"> • Контроль доступу • Службова шина • Послуги в прямому ефірі • Sharepoint
Force.com Salesforce.com	Платформа Force.com: <ul style="list-style-type: none"> • Appex (на основі Java) • VisalForce • Програмування 	Force.com Послуги баз даних	<ul style="list-style-type: none"> • API веб-сервісу • Механізм робочого процесу • Звітність та аналітика • Контроль

	мета		доступу
28мс. Inc.	Саусаліто: XQuery	Інтегр овано в Саусаліто	Бібліотеки функцій Xquery: <ul style="list-style-type: none"> • Аутентифікація • Атом • Пошта

Послуги зберігання переглядаються більш детально у Розділі 1.3.2. Обидві платформи пропонують додаткові послуги, такі як аутентифікація, шини повідомлень тощо, щоб підтримати розробника в програмуванні програми. Ці послуги виходять за рамки даної дипломної роботи.

Force.com, запропонований Salesforce, Inc., застосовує підхід, який дещо відрізняється від Google App Engine та MS Azure. Платформа Force.com бере свій початок від CRM-продукту, що обслуговується як програмне забезпечення, Salesforce.com і, зокрема, призначена для створення традиційних бізнес-додатків [30]. Вся платформа застосовує підхід до розвитку, керований метаданими. Форми, звіти, робочі процеси, привілеї доступу користувачів, ділова логіка, налаштування до таблиць та визначень індексів існують як метадані в Універсальному словнику даних (UDD) Force.com. Force.com підтримує два різні способи створення індивідуальних додатків: декларативний спосіб використання наданої основи програм та програмний підхід використання API веб-сервісу. Мовою програмування, яку підтримує платформа, є APEX, мова, похідна від Java, з операціями маніпулювання даними (наприклад, вставка, оновлення, видалення), операціями контролю транзакцій (setSavepoint, відкат), а також можливістю вбудування Salesforce. мови запитів com (SOQL) та пошуку (SOSL). Наразі відомо, що Salesforce зберігає всі дані одного клієнта / компанії в одній єдиній реляційній базі даних, використовуючи некомпонований макет даних про багатоквартирне житло [30]. Як результат, масштабованість обмежена однією єдиною базою даних.

Поряд з трьома великими платформами на ринку з'являється безліч менших стартапів, що надають платформи різними мовами, такі як Heroku,

Morph, BungeeConnect та 28 мсек. Більшість архітектур наслідують моделі, подібні до таких, як Google і Microsoft. Заслуговує на увагу продукт 28 сантиметрів секунд Sausalito, оскільки він базується на результатах цієї дипломної роботи. Sausalito поєднує сервер додатків та баз даних в один рівень і використовує XQuery як мову програмування. Подібно до Ruby on Rails, платформа забезпечує конфігурацію за загальноприйнятими концепціями та особливо підходить для всіх типів веб-додатків. Sausalito застосовує архітектуру спільного диску, як представлено в главі 3, і повністю розгорнута в інфраструктурі веб-служби Amazon.

Програмне забезпечення як послуга (SaaS)

SaaS - це найвища форма послуг, що надає спеціальне програмне забезпечення через Інтернет. Найбільша перевага для клієнта полягає в тому, що не потрібні попередні інвестиції в сервери або ліцензування програмного забезпечення, а програмне забезпечення повністю підтримується постачальником послуг. З боку провайдера, як правило, одна версія програми розміщується та підтримується для тисяч користувачів, що знижує витрати на розміщення та обслуговування в порівнянні з традиційними моделями програмного забезпечення. Найвидатнішим прикладом SaaS є Salesforce, програмне забезпечення для управління відносинами з клієнтами. Однак більш детальне обговорення SaaS виходить за межі даної тези.

1.2 Додатки веб-баз даних у хмарі

Завдяки сьогоdnішньому вибору можливості створення додатків у хмарі необмежені. Для досягнення однієї і тієї ж мети можна поєднати різні послуги, можливо навіть від різних постачальників. Крім того, за допомогою віртуалізованих машин у хмарі можна створити будь-який вид служби. Хоча немає домовленостей щодо того, як розробляти програми всередині хмари, існують певні рекомендації щодо найкращих практик.

1.2.1 Програми, що використовують PaaS

Найпростіший і, мабуть, найшвидший спосіб розгортання програми всередині хмари - це використання PaaS. Платформа приховує складність базової інфраструктури (наприклад, балансування навантаження, операційна система, стійкість до несправностей, брандмауери) і надає всі послуги, як правило, через бібліотеки, необхідні для побудови програми (наприклад, інструменти для побудови веб-інтерфейсів користувача, зберігання, автентифікація, тестування тощо). Оскільки виклик послуг поза провайдером платформи є досить дорогим щодо затримки, найкращою практикою є використання послуг та інструментів від одного постачальника PaaS. Хоча PaaS пропонує багато переваг, найбільшим недоліком є несумісність між платформами та вимушені обмеження платформи. Крім того, на сьогоднішній день програмне забезпечення PaaS є власним, і жодна система з відкритим кодом не існує,

1.2.2 Програми, що використовують IaaS

Розробники, яким потрібна більша свобода та / або не бажають обмежуватися одним провайдером, можуть безпосередньо використовувати пропозиції IaaS. Вони забезпечують більш прямий доступ до базової інфраструктури. Канонічна форма традиційних веб-архітектур показана на рисунку 1.2. Клієнти підключаються за допомогою веб-браузера через брандмауер до веб-сервера. Веб-сервер відповідає за рендеринг веб-сайтів та взаємодіє з (можливо розподіленою) файловою системою та сервером додатків. Сервер додатків розміщує та виконує логіку програми та взаємодіє з базою даних, а можливо і з файловою системою та іншими зовнішніми системами. Незважаючи на те, що загальна структура схожа на всі веб-програми баз даних, існує величезна різноманітність розширень та стратегій розгортання.

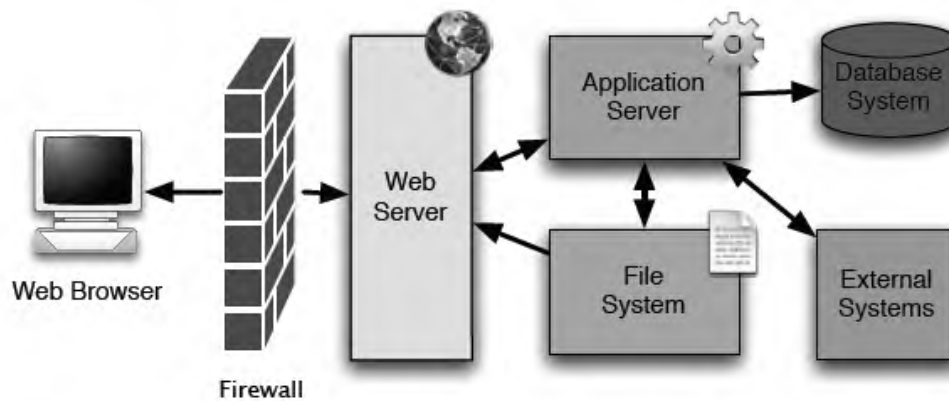


Рисунок 1.2: Канонічна веб-архітектура

хмара. Однак, щоб отримати прибуток від еластичності хмари та моделі ціноутворення "на виплату", не кожна архітектура застосовується однаково. Наприклад, досить популярна монолітна (все-на-одному сервері) архітектура, що уособлюється стеком LAMP (тобто Linux, Apache, MySQL, PHP на одному сервері), не піддається хмарним додаткам. Архітектура представляє не лише одну точку відмови, але оскільки вона обмежена одним екземпляром сервера, її масштабованість у випадку, якщо трафік перевищує потужність машини обмежена.

Найкраща стратегія розгортання веб-додатків, що використовують IaaS, показана на малюнку 1.3. Програма та веб-сервер розгортаються разом на одній віртуальній машині, як правило, за допомогою настроюваного зображення. Як обговорювалося раніше, Vms зазвичай втрачає всі дані, якщо вони аварійно завершують роботу. Таким чином, рівень веб / додатків, як правило, розробляється без громадянства і використовує інші служби для збереження даних сеансу (наприклад, службу зберігання даних або службу баз даних). Служба зберігання даних (наприклад, Amazon S3) зазвичай використовується для великих об'єктів, таких як зображення або відео, тоді як база даних використовується для менших записів. Еластичність архітектури гарантується балансиrom навантаження, який стежить за використанням служб, ініціює нові екземпляри веб-сервера / сервера додатків та балансує навантаження. Балансир навантаження - це або послуга, яка сама може працювати на віртуальній машині,

або послугу, яку пропонує хмарний провайдер. Крім того, брандмауер зазвичай також пропонується як послуга хмарним провайдером.

Щодо розгортання систем баз даних у хмарі, існує два табори думок. Перший табір виступає за встановлення повної транзакційної (кластерної) системи баз даних (наприклад, MySQL, Postgres або Oracle) у хмарі, знову ж таки, за допомогою служби VM. Цей підхід забезпечує комфорт традиційної системи управління базами даних і робить рішення більш автономним від хмарного провайдера, оскільки воно спирається на менш нестандартизовані API. Недоліком є те, що традиційні системи управління базами даних важко масштабувати і не призначені для роботи на віртуальних машинах. Тобто, оптимізатор запитів та управління сховищем припускають, що вони належать виключно апаратному забезпеченню і що помилки досить рідкісні. Все це не тримається всередині хмарного середовища. Крім того, традиційні системи баз даних не розроблені для постійного пристосування до навантаження, що не тільки збільшує вартість,

Другий табір віддає перевагу використанню спеціально розробленого хмарного сховища чи служби баз даних замість повної бази даних транзакцій. Основна архітектура такої послуги, як правило, базується на сховищі ключ-значення, призначеному для масштабованості та стійкості до несправностей. Цей вид служби можна розгорнути самостійно, як правило, використовуючи одну з існуючих систем з відкритим кодом, або пропонується хмарним постачальником. Перевага цього підходу полягає в конструкції системи на відмовостійкість, масштабованість і часто самокерування. Якщо послугою керує хмарний провайдер, це підвищує рівень аутсорсингу. У цьому випадку хмарний постачальник відповідає за моніторинг та підтримку служби зберігання / бази даних. Недоліком є те, що ці послуги часто простіші, ніж повні системи транзакційних баз даних, не пропонуючи ні гарантій ACID, ні повну підтримку SQL. Якщо програма вимагає більшого, її потрібно будувати зверху без особливої підтримки, доступної на сьогодні. Тим не менше, цей підхід стає все більш популярним через свою простоту та чудову масштабованість.

1.3 Системи хмарного зберігання

У цьому розділі розглядаються хмарні системи зберігання даних більш докладно через їх фундаментальну роль у створенні додатків баз даних у хмарі. У той же час, це забезпечує основу для Розділу 2. Тут я використовую ім'я хмара і служба баз даних взаємозамінно, оскільки жодна служба баз даних у хмарі насправді не пропонує такого комфорту, як повноформатна база даних (див. Главу 1.3.2), а з іншого боку, хмарні служби зберігання розширені з більшою функціональністю, що робить їх не простою службою зберігання. Я хотів би наголосити, що я усвідомлюю, що результати, представлені в цій главі, містять лише знімок (червень 2009 р.) Сучасного сучасного рівня обчислювальної техніки.

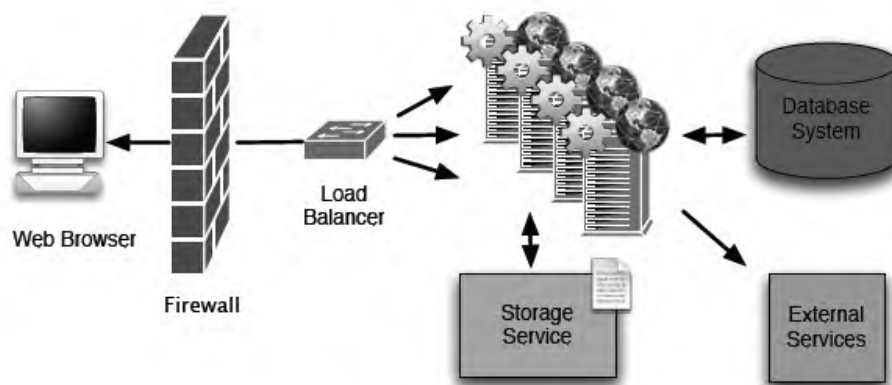


Рисунок 1.3: Архітектура хмарної мережі

Враховуючи успіх AWS та останні пропозиції від Google, цілком ймовірно, що ринок комунальних обчислень та його пропозиції будуть швидко розвиватися. Тим не менше, я вірю, що методи та компроміси, що обговорюються в цій та наступних главах, є фундаментальними і продовжуватимуть залишатися актуальними.

1.3.1 Основи хмарних систем зберігання

У цьому розділі пояснюється важливість теореми CAP для розробки хмарних рішень, перш ніж представити деякі основні інструменти, що використовуються для побудови хмарних сервісів.

Важливість теореми CAP

Для досягнення високої масштабованості при низькій вартості хмарні служби, як правило, є високо розподіленими системами, що працюють на товарному обладнанні. Тут масштабування просто вимагає додавання нового серверного шельфу. На жаль, теорема CAP стверджує, що неможливо одночасно досягти узгодженості, доступності та допуску щодо розділення мережі. Щоб повністю уникнути розділення мережі або, принаймні, зробити це надзвичайно мало ймовірним, можна використовувати окремі сервери або сервери на одній стійці. Обидва рішення не масштабуються і, отже, не підходять для хмарних систем. Крім того, ці рішення також зменшують толерантність до інших збоїв (наприклад, відключення електроенергії або перегрівання). Крім того, використання більш надійних зв'язків між мережами не виключає шансів розділення та значно збільшує вартість. Таким чином, мережових розділів не уникнути, і можна досягти послідовності або доступності. Як результат, хмарній службі потрібно розташуватися десь у просторі дизайну між узгодженістю та доступністю.

Гарантії узгодженості: ACID проти BASE

Сильна узгодженість у контексті систем баз даних зазвичай визначається за допомогою властивостей ACID транзакцій. ACID вимагає, щоб для кожної транзакції мали місце такі атрибути:

Атомність: або всі завдання транзакції виконуються, або жодне.

Узгодженість: Дані залишаються у стабільному стані до початку транзакції та після транзакції.

Ізоляція: Одночасні транзакції призводять до серіалізованого порядку.

Довговічність: Після повідомлення про успіх модифікації транзакції зберігатимуться.

Якщо для узгодженості обрано КИСЛОТУ, це підкреслює узгодженість, одночасно зменшуючи важливість доступності. Потреба в кислоті також передбачає песимістичний погляд, де слід уникати невідповідностей будь-якою ціною. Як наслідок, для досягнення властивостей ACID необхідні складні протоколи, такі як двофазні коміти або протоколи консенсусу, такі як Paxos.

З іншого боку, де доступність важливіша за послідовність, BASE пропонується як протилежність для кислоти. BASE розшифровується як: В основному доступний, М'який стан, Потенційно послідовний. Там, де кислота песимістична і примушує до послідовності в кінці кожної операції, BASE оптимістично сприймає непослідовність. Потенційна узгодженість лише гарантує, що оновлення з часом стануть видимими для всіх клієнтів і що зміни будуть зберігатися, якщо система перейде в стан спокою. На відміну від ACID, можливу послідовність легко досягти і робить систему високодоступною.

Між двома крайніми значеннями BASE та ACID можна знайти цілий ряд моделей узгодженості із спільноти баз даних (наприклад, рівні ізоляції ISO), а також із спільноти розподілених обчислень. Більшість із запропонованих моделей можуть призвести до невідповідностей, але в той же час знизити ймовірність їх виникнення.

Прийоми

Для досягнення відмовостійкості та простоти обслуговування хмарні служби широко використовують розподілені алгоритми, такі як Paxos та розподілені хеш-таблиці. У цьому розділі представлений основний набір інструментів для створення високонадійних та масштабованих хмарних служб, а отже, основні методи порівняння різних хмарних служб. Однак тут основна увага приділяється розподіленим алгоритмам. Стандартні методи баз даних (наприклад, 2-фазний коміт, 3-фазний коміт тощо) вважаються відомими.

Master-Slave / Multi-Master: Найбільш фундаментальним питанням при проектуванні системи є рішення для архітектора-ведучого або мульти-майстра. У моделі master-slave один пристрій або процес має контроль над ресурсом. Кожна зміна ресурсу повинна бути схвалена майстром. Головний керувач, як правило, обирається із групи придатних пристроїв / процесів. У мульти-майстер-моделі управління ресурсом не належить одному процесу; натомість кожен процес / пристрій може модифікувати ресурс. Протокол відповідає за поширення модифікацій даних до решти групи та вирішення можливих конфліктів.

Розподілена хеш-таблиця (DHT): Розподілена хеш-таблиця забезпечує децентралізовану послугу пошуку. У межах DHT зіставлення від ключів до

значень розподіляються по вузлах, часто включаючи деяку надмірність для забезпечення відмовостійкості. Ключовими властивостями DHT є те, що зриви, спричинені приєднанням вузлів або залишків, мінімізовані, як правило, за допомогою послідовного хешування [24], і що жоден вузол не вимагає повної інформації. Реалізації DHT зазвичай відрізняються хеш-методом, який вони застосовують (наприклад, збереження порядку проти випадкового), механізмом балансування навантаження та маршрутизацією до остаточного відображення. Типовим варіантом використання DHT є баланс навантаження та маршрутизація даних через кілька вузлів.

Кворуми: Для оновлення реплік часто використовується протокол кворуму. Система кворуму має три параметри: коефіцієнт реплікації N , кворум читання R і кворум запису W . Запит на читання / запис надсилається всім реплікам N , і кожна репліка, як правило, знаходиться на окремій фізичній машині. Кворум читання R (відповідно кворум запису W) визначає кількість реплік, які повинні успішно брати участь у операції читання (запису). Тобто для успішного зчитування (запису) значення значення має бути зчитане (записане) за допомогою R (W) номерів реплік. Встановлення $R + W > N$ забезпечує постійне читання останнього оновлення. У цій моделі затримка читання / запису диктується найповільнішою з копій читання / запису. З цієї причини R і W зазвичай встановлюються меншими за кількість реплік. Крім того, встановлюючи R і W відповідно, система збалансовує продуктивність читання та запису. Кворуми також визначають доступність та довговічність системи. Наприклад, `smallWinc` збільшує ймовірність втрати даних, і якщо доступно менше вузлів, ніж W or R , читання чи запис більше неможливі. Для систем, де доступність та довговічність важливіші за послідовність, було введено поняття недбалих кворумів. За наявності помилок, недбалі кворуми можуть використовувати будь-який вузол для збереження даних, і вони повторно консолідуються пізніше. Як наслідок, недбалі кворуми також не гарантують зчитування останнього значення, якщо $R + W$ встановлено як більший за N . читати або писати вже неможливо. Для систем, де доступність та довговічність важливіші за послідовність, було введено поняття недбалих кворумів. За наявності помилок, недбалі кворуми можуть використовувати будь-

який вузол для збереження даних, і вони повторно консолідуються пізніше. Як наслідок, недбалі кворуми також не гарантують зчитування останнього значення, якщо $R + W$ встановлено як більший за N . читати або писати вже неможливо. Для систем, де доступність та довговічність важливіші за узгодженість, було введено поняття недбалих кворумів. За наявності помилок, недбалі кворуми можуть використовувати будь-який вузол для збереження даних, і вони повторно консолідуються пізніше. Як наслідок, недбалі кворуми також не гарантують зчитування останнього значення, якщо $R + W$ встановлено як більший за N .

Векторний годинник: Векторний годинник - це список (тобто вектор) пар (клієнт, лічильник), створений для фіксації причинності між різними версіями одного і того ж об'єкта. Таким чином, векторний годинник пов'язаний з кожною версією кожного об'єкта. Кожного разу, коли клієнт оновлює об'єкт, він збільшує свою (клієнтську, лічильникову) пару (наприклад, власний логічний годинник) у векторі на одиницю. Можна визначити, чи суперечать дві версії об'єкта чи причинно-наслідковий порядок, досліджуючи їх векторні годинники. Наведено причинно-наслідковий зв'язок двох версій, якщо кожен лічильник для кожного клієнта вище або дорівнює лічильнику кожного клієнта іншої версії. В іншому випадку існує гілка (тобто конфлікт). Векторні годинники зазвичай використовуються для виявлення конфліктів одночасних оновлень, не вимагаючи контролю узгодженості або централізованої служби [12].

Паксос: Paxos - це консенсус-протокол для мережі ненадійних процесорів. За своєю суттю Паксос вимагає більшості, щоб проголосувати за поточний стан - подібно до кворумів, пояснених вище. Однак Paxos йде далі і може забезпечити високу послідовність, оскільки він здатний відхиляти суперечливі оновлення. Отже, Paxos часто застосовується в архітектурах з декількома майстрами, щоб забезпечити сильну узгодженість - на відміну від простих протоколів кворуму, які зазвичай використовуються в послідовних сценаріях.

Пліткарські протоколи: Протоколи пліток, які також називають епідемічними протоколами, використовуються для передачі інформації всередині системи [11]. Вони працюють подібно до пліток у соціальних мережах, де чутки (тобто інформація) асинхронно поширюються від однієї людини до іншої.

Протоколи пліток особливо підходять для сценаріїв, коли підтримка сучасного перегляду є дорогою або неможливою.

Дерева Меркле: Дерево Меркла або хеш-дерево - це узагальнююча структура даних, де листя є хешами блоків даних (наприклад, сторінок). Вузли, розташовані далі на дереві, є хешами їх відповідних дітей. Хеш-дерева дозволяють швидко визначити, чи змінилися блоки даних, і надалі дозволяють знаходити змінені дані. Таким чином, хеш-дерева зазвичай використовуються для визначення того, чи розходяться репліки одна від одної.

1.3.2 Послуги хмарного зберігання

У цьому розділі подано огляд доступних хмарних служб зберігання, включаючи проекти з відкритим джерелом, які допомагають створювати приватні хмарні рішення.

Послуги комерційного зберігання

Послуги зберігання Amazon: Найвідоміший сервіс зберігання даних - це Amazon S3. S3 - це просте сховище ключ-значення. Система гарантує реплікацію даних у кількох центрах обробки даних, дозволяє сканувати діапазон ключів, але пропонує лише можливі гарантії узгодженості. Таким чином, служби лише обіцяють, що оновлення з часом стануть видимими для всіх клієнтів і що зміни зберігатимуться. Більш досконалі механізми контролю паралельності, такі як транзакції, не підтримуються. Про реалізацію Amazon S3 відомо не так багато. Розділ 2.2 містить докладнішу інформацію про API та інфраструктуру витрат для S3, оскільки я використовую її в частинах наших експериментів.

Поряд із S3, Amazon пропонує Elastic Block Store (EBS). У EBS дані поділяються на обсяги зберігання, і один том може бути змонтований і доступний одночасно рівно одним екземпляром EC2. На відміну від S3, EBS копіюється лише в одному центрі обробки даних і забезпечує узгодженість сеансів.

Внутрішньо Amazon використовує іншу систему під назвою "Динамо". "Динамо" підтримує високі показники оновлення для невеликих об'єктів і тому добре підходить для зберігання кошиків для покупок тощо. Функціонал схожий

на S3, але не підтримує сканування дальності. "Динамо" застосовує багатопрофільну архітектуру, де кожен вузол організований у кільце. Розподілені хеш-таблиці використовуються для полегшення ефективного пошуку, а протокол реплікації та узгодженості базується на кворумах. Несправність вузлів виявляється за допомогою пліток, а дерева Меркла допомагають оновлювати розбіжні репліки. "Динамо" застосовує недбалі кворуми, щоб досягти високої доступності для записів. Єдиною гарантією узгодженості, яку дає система, є остаточна узгодженість, хоча конфігурація кворуму дозволяє оптимізувати типову поведінку (наприклад, монотонність читання-написання).

Служба зберігання Google: Відомо два внутрішні проекти Google: BigTable [6] та Megastore [15]. Останній, Megastore, є, швидше за все, системою, що стоїть за службою зберігання даних AppEngine від Google.

BigTable від Google - це розподілена система зберігання для структурованих даних. Велику таблицю можна розглядати як розріджену, розподілену, стійку багатовимірну відсортовану карту. Карта індексується ключем рядка, ключем стовпця та міткою часу. Жодна схема не нав'язується і не існує інтерфейсу вищого запиту. BigTable використовує архітектуру з одним майстром. Щоб зменшити навантаження на основний пристрій, дані поділяються на так звані планшети, і один планшет обробляється виключно одним веденим (так званий планшетний сервер). Майстер відповідає за (повторне) призначення планшетів на планшетні сервери, за моніторинг, балансування навантаження та певні завдання з обслуговування. Оскільки клієнти BigTable не покладаються на майстер для інформації про місцезнаходження планшета, а запит на читання / запис обробляє сервер планшетного ПК, більшість клієнтів ніколи не спілкуються з майстром.

Chubby, розподілена послуга блокування Google, використовується для вибору ведучого, визначення членства в групах серверів та забезпечення одного планшетного сервера на планшет. Як своє серце, Chubby використовує Paxos для досягнення високої узгодженості між серверами Chubby. Для постійного зберігання даних BigTable покладається на файлову систему Google (GFS) [18]. GFS - це розподілена файлова система для великих файлів даних, які зазвичай

лише додаються і рідко перезаписуються. Таким чином, BigTable зберігає дані на GFS у режимі лише додавання та не перезаписує дані. GFS забезпечує лише розслаблену узгодженість, але оскільки BigTable гарантує один планшетний сервер на планшет, одночасні записи не можуть з'являтися, а на рівні рядків досягається атомність і монотонність. BigTable та Chubby розроблені як єдине рішення центру обробки даних.

Мегастор Google побудований поверх BigTable і дозволяє нав'язувати схеми та простий інтерфейс запитів. Подібно до SimpleDB, мова запитів обмежена, і більш розширені запити (наприклад, приєднання) неможливі. Крім того, Megastore підтримує транзакції з серіалізованими гарантіями всередині групи сутності.¹ Групи сутності повністю визначаються користувачем і не мають обмежень щодо розміру. І все-таки кожна транзакція всередині групи сутностей виконується послідовно. Таким чином, якщо кількість одночасних транзакцій велика, група суб'єктів перетворюється на вузьку горловину. Знову ж таки, ніяких гарантій узгодженості між групами суб'єктів господарювання не надається.

Служба зберігання Yahoo: Відомі дві системи: PNUTS та масштабована платформа даних для невеликих додатків. Перший подібний до великого столу Google. PNUTS застосовує подібну модель даних, а також розділяє дані горизонтально на планшети. На відміну від BigTable, PNUTS призначений для розподілу між кількома центрами обробки даних. Таким чином, PNUTS призначає планшети декільком серверам через межі центру обробки даних.

Кожен планшетний сервер є головним для набору записів із планшетів. Усі оновлення запису переспрямовуються до майстра записів, а потім передаються до інших реплік за допомогою посередника повідомлень Yahoo (YMB). Майстерність запису може мігрувати між репліками залежно від використання і, таким чином, збільшує місцевість для записів. Крім того, PNUTS пропонує API, який дозволяє застосовувати різні рівні узгодженості, такі як можливу послідовність або монотонність.

Друга система Yahoo складається з декількох менших баз даних і забезпечує ¹Також транзакції виконуються в послідовному порядку;

серіалізаційність може бути порушена через лінійні оновлення індексів. сильна консистенція. Вузли / машини організовані в так звані колони з одним колоконтролером у кожному. Вузли в одному кольорі розташовані в одному центрі обробки даних, часто навіть в одній стійці. Вузли в кольорі запускають MySQL та розміщують одну або кілька баз даних користувачів. Контролер solo відповідає за відображення бази даних до вузлів. Крім того, кожна база даних користувачів реплікується всередині solo. Клієнт взаємодіє лише з контролером solo, а контролер пересилає запит до екземпляра MySQL, використовуючи протокол реплікації read-Oncewrite-all. Таким чином, можуть бути надані гарантії кислоти. Крім того, бази даних користувачів тиражуються через колони, використовуючи асинхронний протокол реплікації для аварійного відновлення. Як наслідок, великі збої можуть призвести до втрати даних. Іншим обмеженням, накладеним архітектурою, є те, що ні дані, ні транзакції всередині бази даних не можуть охоплювати більше однієї машини, що передбачає обмеження масштабованості розміру та використання бази даних.

Служба зберігання даних Microsoft: Microsoft пропонує дві послуги: Служба зберігання Azure та База даних Azure SQL. Сховище Windows Azure складається з трьох підслуг: служба BLOB-об'єктів, служба черги, служба таблиці. Послугу BLOB найкраще порівнювати із сховищем ключ-значення для двійкових об'єктів. Послуга черги надає послугу повідомлень, подібну до SQS, а також не гарантує поведінку першого входу / виходу (FIFO). Сервіс планшетів можна розглядати як розширення служби BLOB. Це дозволяє визначати таблиці і навіть підтримує просту мову запитів. У службі зберігання даних Azure дані копіюються в одному центрі обробки даних, і гарантії монотонності надаються на запис, але тут не існує поняття транзакцій для декількох записів. Про реалізацію відомо мало, хоча накладена категоризація даних та обмеження схожі на архітектуру BigTable.

Другою послугою, яку пропонує Microsoft, є база даних SQL Azure. Ця послуга структурована подібно до платформи Yahoo для невеликих додатків, але замість запуску MySQL використовується Microsoft SQL Server. Послуга пропонує повну підтримку транзакцій, спрощену SQL-подібну мову запитів

(поки що не всі функції SQL Server виявляються), але обмежує узгодженість меншим набором записів (тобто 1 або 10 ГБ).

Системи зберігання з відкритим кодом

У цьому розділі подано огляд існуючих систем зберігання з відкритим кодом. Список не є вичерпним, і існує багато інших систем, таких як Токійський кабінет, MongoDB та Рінго. Однак ці системи були обрані, оскільки вони вже є більш стабільними та / або надають деякі цікаві особливості.

Кассандра: Кассандра була розроблена Facebook, щоб забезпечити масштабований зворотний індекс для кожної поштової скриньки користувача. Кассандра намагається поєднати гнучку модель даних BigTable з децентралізованою адміністрацією та завжди доступним для запису підходом Динамо. Для ефективної підтримки зворотного індексу Cassandra підтримує додаткову тривимірну структуру даних, яка дозволяє користувачеві зберігати та запитувати індекси, подібні структурам. Для реплікації та балансування навантаження Кассандра використовує систему кворуму та DHT, подібну до "Динамо".

CouchDB: CouchDB - це база даних документів на основі JSON, написана на Ерлангі. CouchDB може виконувати повнотекстову індексацію збережених документів і підтримує вираження поглядів на дані в JavaScript. CouchDB використовує асинхронну реплікацію на основі однолітків, що дозволяє оновлювати документи на будь-якому рівні. Коли виникають конфлікти розподіленого редагування, детермінований метод використовується для прийняття рішення про виграшну редакцію. Усі інші зміни позначені як суперечливі. Переглянута редакція бере участь у переглядах і надає послідовний погляд. Однак кожна копія все ще може бачити суперечливі версії та має можливість вирішити конфлікт. Механізм транзакцій CouchDB найкраще порівняти з ізоляцією знімків, коли кожна транзакція бачить послідовний знімок. Але на відміну від традиційної ізоляції знімків, конфлікти не призводять до переривань. Натомість,

HBase: HBase - це орієнтоване на стовпці розподілене сховище за зразком BigTable від Google і є частиною проекту Hadoop, платформи MapReduce з

відкритим кодом [10]. Як і BigTable, HBase також покладається на розподілену файлову систему та службу блокування. Файлова система, що надається разом із Hadoop, називається HDFS і схожа на архітектуру Google FileSystem. Служба блокування називається ZooKeeper і використовує з'єднання TCP для забезпечення узгодженості (на відміну від Chubby від Google, який використовує Paxos). Однак вся архітектура HBase, HDFS та ZooKeeper виглядає досить схожою на стек програмного забезпечення Google.

Редіс: Redis - це сховище ключ-значення в пам'яті, що підтримується диском, написане на C. Найцікавішою функцією є модель даних. Redis підтримує не тільки двійкові рядки, цілі числа тощо, але також списки, черги та набори, а також атомні операції вищого рівня над ними (наприклад, push / pop / заміна значень у списку). Redis виконує реплікацію ведучого-підлеглого для надмірності, але без шардування; таким чином, всі дані повинні поміщатися в оперативну пам'ять однієї системи.² Redis зберігає весь набір даних у пам'яті і перевіряє дані час від часу до 2. час на диск. Таким чином, останні оновлення можуть бути втрачені у разі більших збоїв. Реплікація ведучий-підлеглий є синхронною, і кожна зміна, зроблена в ведучому, реплікується якомога швидше на підлеглі.

Скалярія: Scalaris - це пам'ять ключ-значення в пам'яті, написана на Ерлангі. Він використовує модифіковану версію алгоритму Chord для формування DHT і зберігає ключі в лексикографічному порядку, таким чином, дозволяючи запити про діапазон. Дані копіюються за допомогою системи кворуму. Крім того, Scalaris підтримує транзакції через кілька ключів із гарантією ACID, використовуючи розширену версію Paxos. Певним чином, Scalaris поєднує концепцію "Динамо" з "Paxos" і, таким чином, пропонує сильну послідовність.

Проект-Волдеморт: Project-Voldemort - це ще один магазин цінних паперів, розроблений на зразок «Динамо», написаного на Java. Однак він застосовує багатоварову архітектуру, що дозволяє обмінюватися різними компонентами системи. Наприклад, легко обміняти механізм зберігання або метод серіалізації. Здається, система перебуває у надійному стані та працює на LinkedIn.

1.4 XQuery як модель програмування

У цьому розділі наведено огляд існуючих моделей програмування для програм баз даних. Оскільки хмарні програми зазвичай доступні через Інтернет, стандартний інтерфейс користувача - це веб-інтерфейс. Таким чином, у наступному Iwill зосередиться на веб-додатках баз даних.

1.4.1 Огляд моделей програмування

Стандартна модель для веб-додатків все ще є трирівневою архітектурою, де користувальницький інтерфейс, логіка функціональних процесів та доступ до даних розробляються та підтримуються як незалежні модулі, як показано на малюнку 1.4. На різних шарах використовуються різні мови та подання даних. Для клієнта стандартним форматом є HTML або XML, які потім інтерпретуються браузером. Для «програмування» клієнта стандартними мовами є JavaScript або Microsoft ActionScript. Середній рівень надає документи HTML або XML за запитом клієнта. Видатними мовами середнього рівня є Java / JSP, PHP, Ruby on Rails та Python. Якщо додаток працює всередині хмари, середній рівень може бути вже хмарною службою (наприклад, Платформа як послуга) або розміщений на віртуалізованій машині. Зазвичай клієнт взаємодіє із середнім рівнем за допомогою викликів REST, що містять дані JSON або XML. Потім ці дані перетворюються на типи даних хост-мови, такі як об'єкти. Крім того, середній рівень часто є рівнем, що складається з декількох служб, які також спілкуються через XML / JSON. Для збереження та доступу до даних рівень даних відповідає за використання структурованої (наприклад, реляційної) або напівструктурованої (наприклад, XML) моделі даних. Для доступу до декларативних мов даних реляційна) або напівструктурована (наприклад, XML) модель даних. Для доступу до декларативних мов даних реляційна) або напівструктурована (наприклад, XML) модель даних. Для доступу до декларативних мов даних

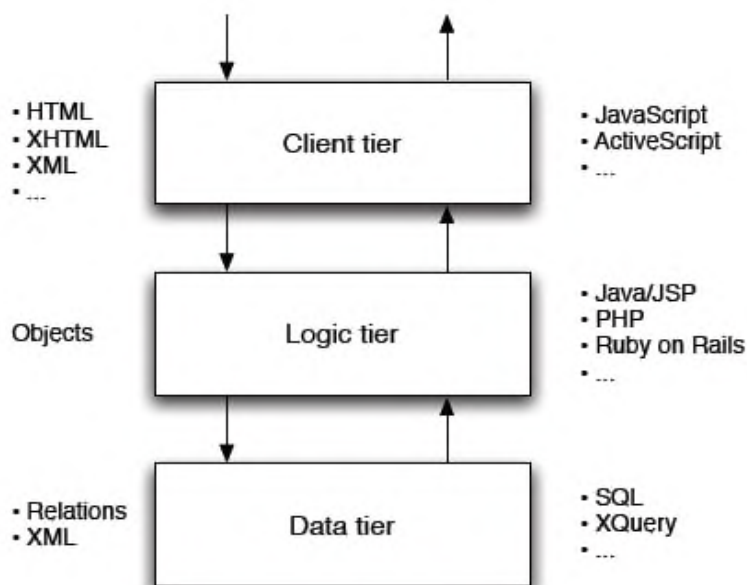


Рисунок 1.4: Шари веб-додатків

такі як SQL або XQuery. Зв'язок між середнім рівнем та рівнем даних зазвичай здійснюється або за допомогою віддалених викликів процедур (RPC), або за допомогою повідомлень XML / JSON.

Однією з найбільших проблем традиційної архітектури є необхідність справлятися з кількома мовами та різними типами даних (наприклад, схема XML проти типів SQL). Крім того, перетворення даних з одного середовища в інше впливає на продуктивність та додає зайвої складності. Підходи, запропоновані для усунення невідповідності між середнім та рівнем даних, поділяються на дві загальні категорії: (1) Покращення мови хоста за допомогою декларативних операторів запитів, щоб йому більше не потрібно було викликати рівень даних; або (2) Покращити мову рівня даних за допомогою розширень для підтримки логіки додатків, не покладаючись на мову середнього рівня. Підхід до вдосконалення мови програмування декларативною мовою для доступу до даних ілюструється такими мовами, як Pascal / R [23], Microsoft LINQ, або певною мірою також Ruby on Rails. Хоча ці розширення зменшують мови, з якими програмісту доводиться мати справу та роблять доступ до даних більш інтегрованим, підхід, як правило, обмежує можливості для оптимізації та не розглядає основну невідповідність між примітивними типами XML Schema та мовою хосту. Розширення мови рівня даних, щоб зробити її незалежною від мови

хосту, як це було зроблено з PL / SQL та SQL PL SQL від IBM, було досить успішним підходом. Сьогодні розширення програмування підтримуються більшістю реалізацій SQL. SQL як мова програмування широко використовувався при створенні багатьох комерційних програм, включаючи salesforce.com та набір програм Oracle. В загальному,

З огляду на поширення даних XML, XQuery нещодавно було запропоновано як альтернативну мову, яка може працювати на всіх рівнях. XQuery - це декларативна мова, спеціально розроблена для XML, хоча вона не обмежується XML. У наступному підрозділі XQuery пояснюється більш докладно як загальна модель програмування для веб-додатків баз даних.

1.4.2 Що таке XQuery?

XQuery - це декларативна мова програмування, яка будується поверх типів даних XML Schema. Отже, XQuery добре підходить для розпаралелювання та уникає невідповідності імпедансу між типами даних XML та типами мови програмування хостингу. З 2007 року W3C рекомендує XQuery 1.0. Наразі майже півсотні реалізацій XQuery рекламуються на веб-сторінках W3C, включаючи реалізації від усіх основних постачальників баз даних та кілька пропозицій з відкритим кодом.

Сама XQuery покладається на кілька стандартів, таких як XML Schema та XPath, і має власну модель даних, яка також використовується спільно з XPath та XSLT. У моделі даних XPath та XQuery (XDM) кожне значення - це впорядкована послідовність з нуля або більше елементів, яка може бути або атомним значенням, або вузлом. Атомне значення - це один з атомних типів, визначений схемою XML, або похідне від одного з них. Вузлом може бути документ, елемент, атрибут, текст, коментар, інструкція обробки або вузол простору імен. Отже, екземпляр моделі даних може містити один або більше XML-документів або фрагментів документів, кожен із яких представлений своїм деревом вузлів. XQuery має кілька заздалегідь визначених функцій та мовних конструкцій, щоб визначити, що і як трансформувати один екземпляр в інший. Найбільш відомою особливістю є вирази FLWOR (вимовляється як "квітка"), що

означає ключові слова "For Let Where Order Return" і є еквівалентом "select from where order by" у SQL. Сам XQuery визначається як перетворення з одного екземпляра моделі даних в інший екземпляр, подібний до функціональної мови програмування. Це також дозволяє підключати кілька XQueries між собою, оскільки кожен результат також є дійсним входом. Вхідні дані поза механізмом XQuery можна вставити, застосовуючи такі функції, як документ або колекція, або посилаючись на зовнішній контекст (попередні змінні). Потім кожен із цих методів повертає екземпляр XDM, який може обробляти XQueries. Основна специфікація XQuery визначає мову запиту. Технічні характеристики «Засоби оновлення XQuery» та «Повнотекстові» XQuery 1.0 та XPath 2.0 додають функціональність для оновлень та повнотекстову підтримку мови. Незважаючи на те, що XQuery (як мова запитів) є цілковитою, суто декларативна концепція ускладнює розробку повних веб-додатків у XQuery. У [20] XQuery поширюється на мову програмування з імперативними концепціями. За допомогою цих розширень стає можливим розробляти повні програми за допомогою XQuery. Оскільки XQuery добре підходить для роботи як на рівні програми, так і на рівні бази даних, можна уникнути навіть невідповідності імпедансу між шарами. Зараз різні пропозиції щодо розширення XQuery переглядаються Робочою групою W3C XQuery. Загальнодоступний робочий проект уже доступний під назвою XQuery Scripting Extension 1.0. XQuery поширюється на мову програмування з імперативними поняттями. За допомогою цих розширень стає можливим розробляти повні програми за допомогою XQuery. Оскільки XQuery добре підходить для запуску як на рівні програми, так і на рівні бази даних, можна уникнути навіть невідповідності імпедансу між шарами. Зараз різні пропозиції щодо розширення XQuery переглядаються Робочою групою W3C XQuery. Публічний робочий проект уже доступний під назвою XQuery Scripting Extension 1.0. XQuery поширюється на мову програмування з імперативними концепціями. За допомогою цих розширень стає можливим розробляти повні програми за допомогою XQuery. Оскільки XQuery добре підходить для запуску як на рівні програми, так і на рівні бази даних, можна уникнути навіть невідповідності імпедансу між шарами. Зараз різні пропозиції щодо розширення XQuery

переглядаються Робочою групою W3C XQuery. Загальнодоступний робочий проект уже доступний під назвою XQuery Scripting Extension 1.0.

1.4.3 XQuery для веб-програм

На даний момент XQuery вже присутній у всіх рівнях веб-програми. На рівні даних XQuery підтримується усіма постачальниками баз даних мер, і існує кілька реалізацій з відкритим кодом. Таким чином, веб-програми вже можуть зберігати дані безпосередньо у форматі XML та отримувати їх за допомогою XQuery. У середньому рівні XQuery має кілька цілей. Одним з яскравих прикладів є трансформація та маршрутизація XML-повідомлень між службами. Інший приклад - інтеграція корпоративної інформації [2]. Третій приклад включає маніпуляції та обробку даних конфігурації, представлених у XML. На рівні клієнта XQuery ще не настільки встановлений, але доступні деякі початкові проекти, щоб зробити XQuery також придатним для використання всередині браузера. Наприклад, за замовчуванням Firefox дозволяє виконувати XPath (підмножину XQuery) всередині JavaScript або плагін XQuery USE ME дозволяє виконувати визначені користувачем XQueries для налаштування веб-сторінок. Однак на середньому рівні, а також на клієнтському рівні XQuery використовується всередині мови хостингу, і тому невідповідність імпедансу все ще існує.

XQuery як повна мова програмування для середнього рівня вперше була досліджена всередині XL-платформи в контексті веб-сервісів. Платформа XL забезпечує віртуальну машину для коду XQuery, декілька розширень мови та структуру, відповідальну за запуск програм XQuery на основі подій (тобто повідомлення веб-служби). Сьогодні найуспішнішим рішенням для XQuery-all є сервер MarkLogic. MarkLogic Server поєднує базу даних XML із сервером додатків XQuery. Таким чином, дані та середній рівень поєднуються на одному сервері. MarkLogic Server особливо добре підходить для програм, орієнтованих на документи, завдяки повнотекстовому пошуку та аналізу тексту, але не обмежується цими сценаріями.

Висновки до розділу

Інтернет спростив надання та споживання вмісту будь-якої форми. Створення веб-сторінки, створення блогу та забезпечення їхнього пошуку для громадськості стали товаром. Тим не менше, надання власної веб-програми / веб-послуги все ще вимагає великих зусиль. Однією з найважливіших проблем є вартість обслуговування служби з ідеальною доступністю 24 х 7 та прийнятною затримкою. Для запуску такої великої послуги, як YouTube, потрібно кілька центрів обробки даних у всьому світі. Запуск служби стає особливо складним та дорогим, якщо послуга успішна: успіх в Інтернеті може вбити! Для подолання цих проблем було запропоновано обчислювальні програми (відомі як хмарні обчислення) як новий спосіб роботи з послугами в Інтернеті [27].

Завдання хмарних обчислень - за допомогою спеціалізованих розподільників - забезпечити основні інгредієнти, такі як сховище, процесори та пропускна здатність мережі, як товар із низькою вартістю одиниці. Наприклад, у випадку зберігання службових програм користувачам не потрібно турбуватися про масштабованість, оскільки надане сховище практично нескінченне. Крім того, обчислювальні програми забезпечують повну доступність, тобто користувачі можуть читати та записувати дані в будь-який час, ніколи не блокуючись. Час відповіді (практично) постійний і не залежить від кількості одночасних користувачів, розміру бази даних або будь-якого іншого системного параметра. Крім того, користувачам не потрібно турбуватися про резервні копії. Якщо компоненти виходять з ладу, відповідальність постачальника комунальних послуг - замінити їх і тим часом зробити доступними дані за допомогою реплік. Ще однією важливою причиною побудови нових послуг на основі хмарних обчислень є те, що постачальники послуг платять лише за те, що отримують, тобто платять за використання. Попередні інвестиції не потрібні, а вартість зростає лінійно та передбачувано із використанням. Хоча іноді його розглядають як ажіотаж, головним успіхом хмарних обчислень є не технологічний, а економічний. Хмарні обчислення дозволяють компаніям передавати ІТ-інфраструктуру на аутсорсинг і, таким чином, отримувати прибуток від економії

масштабу та ефекту впливу аутсорсингу. Хоча переваги для створення додатків у хмарі є переконливими, вони мають певні обмеження. На сьогоднішній день немає єдиної думки щодо хмарних сервісів. Таким чином, різні провайдери пропонують різну функціональність та інтерфейси, що ускладнює перенесення програм від одного провайдера до іншого. Крім того, системи жертвують функціональністю та послідовністю, щоб забезпечити краще масштабування та доступність. Якщо потрібна більша кількість функціональних можливостей та / або послідовність, її слід будувати зверху. Хоча деякі хмарні провайдери пропонують рекомендації щодо найкращих практик щодо створення додатків у хмарі, нові компроміси - особливо для програм, які можуть вимагати більш суворих гарантій узгодженості (наприклад, додатків баз даних) взагалі не розглядаються.

РОЗДІЛ 2

ПІДХІД ДО РОЗРОБКИ ОНЛАЙНОВОГО РЕПОЗИТОРІЮ ЧЕРЕЗ ПАРАДИГМУ ОБЛАЧНОЇ ІНФРАСТРУКТУРИ

Розробка програмного забезпечення веде до залучення сукупності аспектів, що впливають на всі етапи процесу розробки програмного забезпечення, як на організаційно-економічну, так і на мови програмування та середовища розробки. З кожним новим кроком до еволюції розвитку виникають нові фактори успіху, а отже, і нові проблеми зростають.

2.1 Мотивація

Хмарні обчислення призвели до нових рішень для зберігання та обробки даних у хмарі (як показано в розділі 1.1). Прикладами є Amazon S3, BigTable від Google або PNUTS Yahoo. Порівняно з традиційними системами транзакційних баз даних, основною перевагою цих нових служб зберігання є їх еластичність в умовах мінливих умов. Наприклад, щоб динамічно адаптуватися до навантаження, хмарні провайдери автоматично розподіляють і розподіляють ресурси (тобто обчислювальні вузли) на льоту, пропонуючи для виставлення рахунків обчислювальну модель pay-as-you-go. Однак з точки зору функціональності ці нові служби зберігання даних далекі від того, що пропонують системи баз даних.

Однією з найважливіших відмінностей між хмарними службами зберігання даних порівняно із традиційними системами баз даних транзакцій є надані гарантії узгодженості. Щоб забезпечити стійкість до несправностей та високу доступність, більшість постачальників копіюють дані в одному або навіть у кількох центрах обробки даних. Дотримуючись теореми CAP,

Кафедра КСУ				НАУ 21 08 21 000 ПЗ							
Виконав	Орнатська Є.Д.			Підхід до розробки онлайнного репозиторію через парадигму програмних екосистем	Літера		Аркуш		Аркушів		
Керівник	Росінська Г.П.				Д		41		72		
Консульт.					СП 501Бз 123						
Норм. контр.	Тупота Є.В.										
Зав. Каф.	Литвиненко О.Є.										

неможливо забезпечити спільну доступність та сильну узгодженість (як визначено властивостями ACID) за наявності відмов мережі. Отже, більшість хмарних провайдерів жертвують сильним узгодженням для наявності та пропонують лише деякі слабкі форми узгодженості (наприклад, S3 Amazon гарантує лише можливу узгодженість). Якщо потрібен більш високий рівень узгодженості, його потрібно будувати зверху. Крім того, можливості обробки запитів далекі від повної підтримки SQL. Більшість систем пропонують пошук ключів та діапазону ключів (див. Розділ 1.3.2). Небагато систем підтримують вдосконалені фільтри і ще менше підтримують спрощену мову запитів з можливістю замовлення та / або підрахунку.

За винятком служб даних Azure SQL, жодна доступна загальнодоступна служба не пропонує об'єднання або більш розширені агрегати. Знову ж таки, будь-яка додаткова функціональність повинна бути побудована зверху.

Незважаючи на те, що можна встановлювати традиційні системи баз даних транзакцій у хмарі (наприклад, MySQL, Oracle, IBM DB2), ці рішення не масштабуються та адаптуються до навантаження так, як це роблять служби зберігання, такі як S3 (див. Також Розділ 1.2.2). Крім того, традиційні системи баз даних не надаються як послуга, тому їх потрібно встановлювати, налаштовувати та контролювати, таким чином виключаючи переваги передачі бази даних на аутсорсинг (див. Також Розділ 1.2.2). Мета цього розділу - дослідити, як веб-програми баз даних (у будь-якому масштабі) можуть бути реалізовані поверх служб зберігання, таких як S3, подібних до архітектури спільного диска. У цій главі представлені різні протоколи для зберігання, читання та оновлення об'єктів та індексів за допомогою служби зберігання.

2.2 Довідковий хмарний API для додатків баз даних

Цей розділ описує API, який абстрагується від деталей хмарних служб, таких як пропоновані Amazon, Google, Microsoft та іншими постачальниками послуг, представлених у розділі 1.1. Усі протоколи та методи, представлені для

розробки веб-застосунків баз даних у решті цього розділу, базуються на викликах цього API. Цей API визначає не тільки інтерфейси, а й гарантії, які повинні надавати різні послуги. Розділ 2.6 ілюструє використання AWS, як цей API може бути реалізований.

2.3.1 Послуги зберігання

Найважливіший будівельний блок - надійний магазин. Далі наведено перелік методів, які легко запровадити за допомогою сучасних хмарних служб (наприклад, AWS та Google), тим самим абстрагуючись від специфіки постачальника. Цей список можна розглядати як найбільший спільний діляник для всіх поточних постачальників хмарних послуг та мінімум, необхідний для створення додатків із підтримкою стану:

поставити(uri як рядок, корисне навантаження як двійкове, метадані як пари ключ-значення) як void: зберігає елемент (або об'єкт), ідентифікований даним URI. Елемент складається з корисного навантаження, метаданих та мітки часу останнього оновлення. Якщо URI вже існує, елемент перезаписується без попередження; в іншому випадку створюється новий елемент.

отримати(uri як рядок) як елемент: отримує елемент (корисне навантаження, метадані та мітку часу), пов'язаний з URI.

get-метадані(uri як рядок) як пари ключ-значення: повертає метадані елемента.

getIfModifiedSince(uri як рядок, позначка часу як час) як елемент: Повертає елемент, пов'язаний з URI, лише якщо мітка часу елемента перевищує позначку часу, передану як параметр виклику.

елементи списку(uriPrefix як рядок) як елементи: Перераховує всі елементи, чиї URI відповідають даному префіксу URI.

видалити(uri як рядок) як void: видаляє елемент.

Для стислості Ido не вказує всі коди помилок. Крім того, Ido не вказує операції з надання та видалення прав доступу, оскільки впровадження безпеки хмарних служб виходить за межі даної тези.

Що стосується узгодженості, я очікую, що служба хмарного сховища підтримає можливу узгодженість з консолідацією на основі часу. Тобто до кожного оновлення додається позначка часу, і якщо потрібно оновити два оновлення, виграє найвища позначка часу. Однак я не припускаю глобальний годинник, і кілька записів за дуже короткий проміжок часу можуть зазнати впливу годинникових гвинтів всередині системи. Знову ж таки, остаточна узгодженість (із вирішенням конфліктів за останніми оновленнями та перемогами), здається, є стандартом, який пропонують сьогодні більшість хмарних служб, оскільки він дозволяє масштабувати та забезпечує 100-відсоткову доступність за низькою вартістю.

2.3.2 Бронювання машини

Розгортання користувацьких програм у хмарі вимагає розгортання та запуску спеціального коду програми в хмарі. Сучасна тенденція полягає у наданні хостинг-платформи для певної мови, наприклад, AppEngine від Google, або можливості запуску віртуалізованої машини, як це зроблено в Amazon EC2. Платформи хостингу часто обмежені певною мовою програмування і приховують подробиці про те, як виконується код. Оскільки віртуалізовані машини є більш загальними, я пропоную API дозволити запускати та вимикати машини наступним чином:

почати(machineImage як рядок) як machineDescription: запускає дане зображення та повертає інформацію про віртуальну машину, таку як її ідентифікатор та загальнодоступне ім'я DNS.

Стоп(ідентифікатор машини як рядок) як void: Зупиняє віртуальну машину з ідентифікатором машини

Для стислості способи створення власних зображень не описані.

2.3.3 Прості черги

Черги є важливим будівельним блоком для створення веб-застосунків баз даних у хмарі та пропонуються декількома хмарними провайдерами (наприклад,

Amazon та Google) як частина їх інфраструктури. Черга повинна підтримувати такі операції (знову ж таки, коди помилок не вказані для стислості):

`createQueue(uri як рядок)` як `void`: Створює нову чергу із заданим URI.

`deleteQueue(uri як рядок)` як `void`: Видаляє чергу.

`listQueues()` у вигляді рядків: повертає URI як рядок усіх черг.

`відправити повідомлення(uri як рядок, корисне навантаження як двійкове)` як ціле число: надсилає повідомлення з корисним навантаженням як вміст до черги та повертає `MessageID`. `MessageID` - це ціле число (не обов'язково впорядковане за часом).

`receiveMessage(uri як рядок, N як ціле число)` як повідомлення: Отримує N повідомлень із черги. Якщо доступно менше N повідомлень, повертається якомога більше повідомлень. Звичайно, повідомлення повертається з його ідентифікатором повідомлення та корисним навантаженням.

`deleteMessage(uri як рядок, messageID як ціле число)` як `void`: видаляє повідомлення (ідентифіковане `MessageID`) з черги (ідентифіковане за його URI).

Черги повинні бути надійними і ніколи не втрачати повідомлення.¹ Прості черги не дають жодних гарантій FIFO. Отже, дзвінок `receiveMessage` може повернути друге повідомлення без першого повідомлення. Крім того, Прості черги не гарантують доступності всіх повідомлень у будь-який час; тобто, якщо в черзі N повідомлень, можливо, виклик `receiveMessage` із запитом на N повідомлень повертає менше N повідомлень. Таким чином, прості черги можуть працювати навіть за наявності мережевого розділення і, отже, можуть забезпечувати найвищу доступність (тобто вони жертвують властивістю узгодженості для наявності). Усі протоколи, побудовані поверх цих простих черг, повинні поважати можливі невідповідності. Як стане зрозуміло в 2.5, продуктивність протоколу покращується, чим краще черга відповідає принципу FIFO і чим більше повідомлень повертає черга як частина виклику `receiveMessage`. Деякі протоколи вимагають більш суворих гарантій; ці протоколи повинні бути реалізовані поверх `Advanced Queue`.

2.3.4 Розширені черги

Порівняно з простими чергами, розширені черги забезпечують більші гарантії. Зокрема, розширені черги можуть надавати користувачеві всі успішно передані повідомлення в будь-який момент і завжди повертати повідомлення в тому ж порядку. Тобто, в той момент, коли запит на відправлення до черги успішно повертається, повідомлення зберігається і може бути отримане шляхом отримання запитів на отримання. Усі повідомлення вводяться в загальний порядок, але збільшення порядку між двома повідомленнями $m1$ і $m2$ гарантується лише в тому випадку, якщо $m2$ відправляється після успішного повернення запиту надсилання для $m1$. Як наслідок додаткових гарантій, очікується, що запити до цих черг будуть дорожчими, а розширені черги можуть тимчасово бути недоступними через теорему CAP. Додатковою відмінністю між розширеними чергами та простими чергами є наявність операторів у розширеній черзі, що дозволяє фільтрувати повідомлення. Розширені черги надають спосіб приєднати визначений користувачем ключ (на додаток до MessageID) до кожного повідомлення. Ця клавіша дозволяє додатково фільтрувати повідомлення. Така послуга потрібна не тільки для вдосконалених протоколів, але й корисна в багатьох інших сценаріях. Наприклад, розширені черги можуть використовуватися для підключення виробників та споживачів, коли порядок повідомлень не можна ігнорувати (наприклад, при обробці подій, моніторингу тощо) або він може використовуватися як надійний механізм обміну повідомленнями між компонентами. Знову ж таки, деталі нашої реалізації розширених черг поверх AWS наведено у розділі 2.6. API розширених черг такий: Розширені черги надають спосіб приєднати визначений користувачем ключ (на додаток до MessageID) до кожного повідомлення. Ця клавіша дозволяє додатково фільтрувати повідомлення. Така послуга потрібна не тільки для вдосконалених протоколів, але й корисна в багатьох інших сценаріях. Наприклад, розширені черги можуть використовуватися для підключення виробників та споживачів, коли порядок повідомлень не можна ігнорувати (наприклад, при обробці подій, моніторингу тощо) або він може використовуватися як надійний механізм обміну повідомленнями між компонентами. Знову ж таки, деталі нашої реалізації розширених черг поверх AWS наведено у розділі 2.6. API розширених

черг такий: Розширені черги надають спосіб приєднати визначений користувачем ключ (на додаток до MessageID) до кожного повідомлення. Ця клавіша дозволяє додатково фільтрувати повідомлення. Така послуга потрібна не тільки для вдосконалених протоколів, але й корисна в багатьох інших сценаріях. Наприклад, розширені черги можуть використовуватися для підключення виробників та споживачів, коли порядок повідомлень не можна ігнорувати (наприклад, при обробці подій, моніторингу тощо) або він може використовуватися як надійний механізм обміну повідомленнями між компонентами. Знову ж таки, деталі нашої реалізації розширених черг поверх AWS наведено у розділі 2.6. API розширених черг такий: але також корисний у багатьох інших сценаріях. Наприклад, розширені черги можуть використовуватися для підключення виробників та споживачів, коли порядок повідомлень не можна ігнорувати (наприклад, при обробці подій, моніторингу тощо) або він може використовуватися як надійний механізм обміну повідомленнями між компонентами. Знову ж таки, деталі нашої реалізації розширених черг поверх AWS наведено у розділі 2.6. API розширених черг такий:

`createAdvancedQueue(uri як рядок)` як void: Створює нову розширену чергу з вказаним URI.

`deleteAdvancedQueue(uri як рядок)` як void: Видаляє чергу.

`відправити повідомлення(uri як рядок, корисне навантаження як двійкове, ключ як ціле число)` як ціле число: надсилає повідомлення з корисним навантаженням як вміст до черги і повертає MessageID. MessageID - це ціле число, яке впорядковується відповідно до часу надходження повідомлення в чергу (повідомлення, отримані раніше, мають нижчий MessageID). Крім того,

розширені черги підтримують приєднання визначених користувачем ключів до повідомлень (представлених як цілі числа) для подальшої фільтрації.

`receiveMessage(uri як рядок, N як ціле число, messageIdGreaterThan як ціле число)` як повідомлення: Повертає верхні N повідомлень, чиї `MessageID` перевищує “`messageIdGreaterThan`”. Якщо таких повідомлень менше ніж N, `Advanced`. Черга повертає всі відповідні повідомлення. На відміну від простих черг, метод `receiveMessage` розширених черг дотримується принципу FIFO.

`receiveMessage(uri як рядок, N як ціле число, keyGreaterThan як ціле число, ключ-LessThan як ціле число)` як повідомлення: Отримує верхні N повідомлень, ключ яких відповідає вказаному діапазону значень ключа. З точки зору FIFO та гарантій повноти, ця версія операції `receiveMessage` поводитьсь точно так само, як операція `receiveMessage`, яка фільтрується на `MessageID`.

`receiveMessage(uri як рядок, N як ціле число, старше ніж за секунди)` як повідомлення: Отримує верхні N повідомлень у черзі, які старші за старих ніж секунди. Ця функція часто потрібна для відновлення від сценаріїв відмов (наприклад, повідомлення не були оброблені вчасно).

`deleteMessage(uri як рядок, messageId як ціле число)` як `void`: видаляє повідомлення (ідентифіковане `MessageID`) з черги (ідентифіковане за його URI).

`deleteMessages(uri як рядок, keyLessThan як ціле число, keyGreaterThan як ціле число)` як `void`: видаляє всі повідомлення, ключ яких знаходиться у вказаному діапазоні.

2.3.5 Служба блокування

Служба блокування реалізує централізовану службу для відстеження блокування читання та запису. Клієнт (ідентифікований ідентифікатором) може отримати спільний блокування ресурсу, вказаного URI. Крім того, користувачі можуть придбати ексклюзивні замки. Згідно з загальноприйнятою думкою, кілька різних клієнтів можуть одночасно тримати загальні замки, тоді як ексклюзивні замки можуть утримуватися лише одним клієнтом і виключають надання спільних замків. Замки надаються лише на певний термін. Термін дії блокування закінчується після вказаного тайм-ауту, забезпечує жвавість системи

на випадок аварії клієнта, що тримає замок. Якщо замок потрібно тримати довше, його можна отримати повторно до закінчення строку, використовуючи той самий ідентифікатор. На щастя, ексклюзивні блокування можна легко впровадити поверх SQS, як показано в Розділі 3.7, але реалізуючи це як спеціальну хмарну службу,

API служби блокування виконує такі дії:

`setTimeout(префікс як рядок, час очікування як ціле число)` як `void`:
Встановлює час очікування блокування для ресурсів, визначених певним префіксом URI.

`придбатиXLock(uri як рядок, id як рядок, час очікування як ціле число)` як логічне значення: отримує ексклюзивний замок. Повертає `true`, якщо блокування було надано, а `false` - інакше.

`придбатиSlock(uri як рядок, ідентифікатор як рядок, час очікування як ціле число)` як логічне значення: отримує спільний замок. Повертає `true`, якщо блокування було надано, а `false` - інакше.

`releaseLock(uri як рядок, ідентифікатор як рядок)` як логічне значення: звільняє блокування.

Що стосується надійності, можливо, послуга блокування не вдається. Можливо також, що служба блокування втрачає свій стан як наслідок такої несправності. Якщо служба блокування відновлюється після такої несправності, вона відмовляє у всіх запитах `придбатиXLock` та `придбатиSlock` протягом максимального періоду очікування, щоб гарантувати, що всі клієнти, які тримають замки, можуть закінчити свою роботу до тих пір, поки вони утримують замки.

Таким чином, доступність служби блокування зменшується: вона не доступна ні під час розділення мережі, ні навіть під час відмов. Тоді як останнього можна уникнути, служба завжди повинна забезпечувати високу узгодженість і, отже, не завжди може бути доступною, якщо можливі розділи мережі. Однак більшість протоколів, представлених у розділах 2.5, передбачають, що недоступність служби блокування відповідає відсутності

можливості вимагати блокування. Це дозволяє продовжувати роботу залежно від протоколу без додаткових недоліків.

2.3.6 Розширені лічильники

Покращена послуга лічильника - це спеціальна послуга, призначена для реалізації протоколу загальної ізоляції знімків (Розділ 2.5). Як випливає з назви, розширена служба лічильника реалізує лічильники, які збільшуються з кожним додатковим викликом. Кожен лічильник ідентифікується за допомогою URI. Як особливість, розширена послуга лічильника дозволяє перевіряти значення лічильника, а також отримувати найвище підтверджене значення лічильника. Якщо явно не перевірено, значення лічильника автоматично перевіряються після певного періоду очікування. Як показано в розділі 2.5.4, ця функція перевірки важлива для реалізації протоколу коміту ізоляції знімків. Таким чином, API містить такі операції:

`setTimeout`(префікс як рядок, час очікування як ціле число) як `void`: встановлює час очікування всіх лічильників для певного префіксу URI.

`збільшення`(uri як рядок) як ціле число: Збільшує лічильник, ідентифікований параметром uri, і повертає поточне значення лічильника (включаючи епічне число). Якщо лічильник з цим URI не існує, створюється новий лічильник, який ініціалізується до 0.

`перевірити`(uri як рядок, значення як ціле число) як `void`: перевіряє значення лічильника. Якщо не викликається явно, значення лічильника перевіряється автоматично з урахуванням інтервалу часу очікування після виклику збільшення, який створив це значення лічильника.

`getHighestValidatedValue`(uri як рядок) як ціле число: повертає найвище підтверджене значення лічильника.

Як і служба блокування, усі протоколи повинні бути розроблені з урахуванням того, що вдосконалена служба лічильника може вийти з ладу в будь-який час. Коли розширена послуга лічильника відновлюється, вона скидає всі лічильники до 0 з новим вищим епічним числом. Однак після перезапуску розширена служба лічильника відхиляє всі запити (повертає помилку) протягом

максимального періоду очікування всіх лічильників. Отже, останнє епічне число та максимальний період очікування всіх лічильників повинні зберігатися постійно та відновлюватися (наприклад, у таких службах, як S3).

Як і послуга блокування, ця послуга також має знижену доступність. Однак, на відміну від служби блокування, недоступність, як правило, не можна ігнорувати, а отже, також зменшує доступність усіх систем залежно від неї. Отже, реалізації, що використовують цю послугу, повинні знати про її доступність.

2.3.7 Індексція

Послуга індексації корисна в ситуаціях, коли самокеровані індекси, що зберігаються в службі зберігання, занадто дорогі з точки зору вартості та продуктивності. Це часто трапляється, коли весь стек додатків повинен розміщуватися на рівні клієнта, а не на сервері додатків, або для рідко використовуваних додатків з величезними обсягами даних.

Багато хмарних провайдерів пропонують послуги, які можна використовувати як індекс. SimpleDB від Amazon розроблений як індекс і підтримує запити щодо точок і діапазонів. Інші постачальники часто пропонують послуги з обмеженими базами даних. Наприклад, SQL Azure від Microsoft надає службу баз даних у хмарі, але лише до 10 Гб. Google DataStore не має обмежень щодо розміру даних, але синхронізує кожен запит на запис всередині однієї групи сутностей. Хоча через їх обмеження часто не бажано використовувати ці служби безпосередньо як базу даних, ці служби можна легко використовувати для надання послуги індексації, де обмеження не відіграють важливої ролі (див. Розділ 2.6.7).

Вимоги до служби індексування подібні до вимог служби зберігання даних у підрозділі 2.3.1. Ключі повинні зберігатися разом із корисним навантаженням і бути доступними за допомогою ключа. Найбільша відмінність полягає в тому, що ключ не повинен бути унікальним, що корисне навантаження досить мало і що можливі отримання запитів з ключем та діапазоном ключів.

вставити(ключ як рядок, корисне навантаження як рядок) як void: зберігає корисне навантаження, ідентифіковане даним ключем. Ключ не повинен бути унікальним.

отримати(ключ як рядок) як встановлений (пара (ключ як рядок, корисне навантаження як рядок)): Отримує всі пари ключ / значення, пов'язані з ключем.

отримати(startkey як рядок, endkey як рядок) як встановлений (пара (ключ як рядок, корисне навантаження як рядок)): Отримує всі пари ключ / значення для всіх ключів у діапазоні між startkey та endkey (включно).

видалити(ключ як рядок, корисне навантаження як рядок) як void: видаляє першу пару ключ / значення із відповідними ключем та корисним навантаженням.

Що стосується узгодженості, я очікую на можливу узгодженість або на прочитане, залежно від бажаної загальної гарантії узгодженості і, отже, застосованого протоколу. Ця послуга є необов'язковою для всіх протоколів, представлених у цій дипломній роботі. Якщо хмарний постачальник не пропонує послугу індексації або забезпечує нижчий рівень узгодженості, ніж потрібно, завжди можна використовувати самокеровані індекси.

2.4 Переглянута архітектура бази даних

Як згадувалось у розділі 2.2.1, хмарні обчислення обіцяють нескінченну масштабованість, доступність та пропускну здатність - з точки зору малих та середніх компаній. Цей розділ показує, що багато методів підручника для реалізації таблиць, сторінок, В-дерев та журналювання можуть бути застосовані для реалізації бази даних поверх Cloud API розділу 2.3. Мета цього розділу - висвітлити спільність між дисковою та хмарною системою баз даних. Розділ 3.5 висвітлює також існуючі відмінності.

2.4.1 Архітектура клієнт-сервер

На рисунку 2.1 представлена запропонована архітектура бази даних, реалізована поверх API Cloud Storage, описана в розділі 2.3.1. Ця архітектура має багато спільного з розподіленою системою баз даних на спільному диску.

Одиницею передачі та буферизації є сторінка. Різниця полягає в тому, що сторінки постійно зберігаються в хмарі, а не на диску, який безпосередньо контролюється системою баз даних. У цій архітектурі сторінки реалізовані як елементи в API служби зберігання даних розділу 2.3.1.

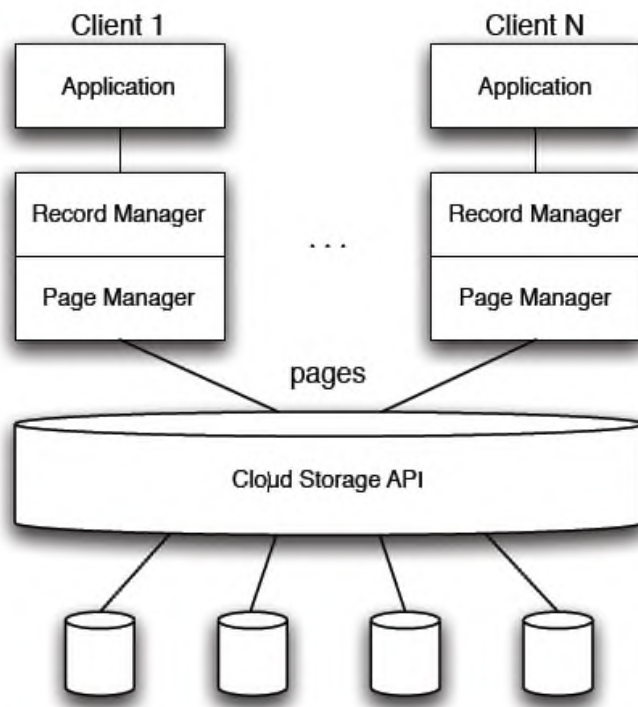


Рисунок 2.1: Архітектура спільного диска

Як і в традиційних системах баз даних, сторінка містить набір записів або записів індексу. Дотримуючись загальної термінології БД, Irefer - записи як потік змінних розмірів, який не може бути більшим за розмір сторінки. Записи можуть бути реляційними кортежами або елементами та документами XML. Краплі (для записів, що перевищують розмір сторінки) можна зберігати безпосередньо в службі зберігання або за допомогою методів, усі ці методи застосовуються прямолінійно, і тому краплі більше не обговорюється в цій главі.

Усередині клієнта є безліч компонентів, що підтримують додаток. Ця робота зосереджена на двох найнижчих шарах; тобто менеджери записів та сторінок. На всі інші рівні (наприклад, процесор запитів) використання хмарних служб не впливає, і з цієї причини вважається частиною програми. Менеджер сторінок координує запити на читання та запис у Службу зберігання та

буферизує сторінки в локальній основній пам'яті або на диску. Менеджер записів забезпечує орієнтований на записи інтерфейс, організовує записи на сторінках та здійснює управління вільним простором для створення нових записів. Програми взаємодіють лише з диспетчером записів, використовуючи тим самим інтерфейс, описаний у наступному підрозділі. Знову ж таки, для цілей цього дослідження процесор запитів (оптимізатор та час роботи) є частиною програми.

Протягом усієї роботи я використовую термін клієнт для позначення програмних артефактів, які отримують сторінки та записують сторінки в Службу хмарного зберігання. Цікавим є питання, які частини стеку клієнтських програм повинні працювати на машинах, що надаються постачальником хмарних послуг (наприклад, машини EC2), а які частини стеку програм повинні працювати на машинах, що надаються кінцевими користувачами (наприклад, ПК, ноутбуки, мобільні телефони тощо). Справді, можна реалізувати архітектуру веб-баз даних, використовуючи цю архітектуру, не використовуючи машин від постачальника хмарних послуг.

Пов'язане питання стосується реалізації індексів: Одним із варіантів є використання SimpleDB (або супутніх служб) для реалізації індексації. Альтернативою є впровадження B-Trees поверх хмарних служб таким же чином, як традиційні системи баз даних реалізують B-Trees поверх дисків

Незалежно від того, працює стек клієнтських програм на машинах користувачів або, скажімо, на машинах EC2, архітектура Рисунок 2.1 розроблена для підтримки тисяч, якщо не мільйонів клієнтів. Як результат, усі протоколи повинні бути розроблені таким чином, щоб будь-який клієнт міг вийти з ладу в будь-який час і, можливо, ніколи не відновитись після відмови. Як результат, клієнти не мають громадянства. Вони можуть кешувати дані із хмари, але найгірше, що може статися, якщо клієнт зазнає невдачі, це те, що вся робота цього клієнта втрачена.

В решті цього розділу більш докладно описані менеджер записів, менеджер сторінок, реалізація індексів (B-Tree) та ведення журналу. Управління метаданими, таке як управління каталогом, який реєструє всі колекції та індекси, у цій дисертації не обговорюється. Це реалізувати просто, оскільки вся

інформація зберігається в самій базі даних так само, як традиційні (реляційні) бази даних зберігають каталог всередині деяких прихованих таблиць.

2.4.2 Менеджер записів

Менеджер записів управляє записами (наприклад, реляційними кортежами). Кожен запис пов'язаний з колекцією (див. Нижче). Запис складається з ключа та даних корисного навантаження. Ключ однозначно ідентифікує запис у його колекції. Дані як ключового, так і корисного навантаження є бітестами довільної довжини; єдине обмеження полягає в тому, що розмір цілого запису повинен бути меншим за розмір сторінки. (Як вже згадувалося раніше, реалізація Blobs не розглядається в цій дисертації.)

Фізично кожен запис зберігається рівно на одній сторінці, яка, в свою чергу, зберігається як один елемент за допомогою API Cloud Store. Логічно, що кожен запис є частиною колекції (наприклад, таблиці). У нашому впровадженні колекція ідентифікується за допомогою URI. Усі сторінки колекції використовують URI колекції як префікс. Менеджер записів надає функції для створення нових записів, читання записів, оновлення записів та сканування колекцій.

Створити (ключ, корисне навантаження, `uri`): Створює новий запис у колекції, визначений `uri`. Якщо ключ не унікальний, то `create` повертає помилку (якщо помилку можна виявити негайно) або ігнорує запит. Для того, щоб реалізувати ключі, які гарантовано будуть унікальними в розподіленій системі, `Iuse uuids`, згенеровані апаратним забезпеченням клієнта в нашій реалізації. `Uuid` - це 128-бітний ідентифікатор, що складається з MAC-адреси користувача, часу та деякого випадкового числа, що робить зіткнення майже неможливим.

Існує багато альтернативних способів реалізації управління вільним простором [26], і всі вони застосовні в цьому контексті. У нашій поточній реалізації я використовую таблицю вільного місця, яка зберігається разом із інформацією про індекс індексу первинного ключа в службі зберігання. Це дозволяє використовувати той самий тип вільного управління простором для B-дерев на стороні клієнта, а також для служб індексації. За допомогою таблиці

вільного простору оцінюється найкраща сторінка для запису. Це лише оцінка, оскільки інформація про розмір сторінки може бути застарілою завдяки оновленням записів. В ідеалі присвоєння запису сторінці виконується лише один раз. Однак, оскільки записи можуть зростати, через поля змінної довжини може знадобитися переміщувати записи між сторінками. Щоб уникнути цих переміщень, я зарезервую певний відсоток місця для оновлень, подібний до параметра PCTFREE БД Oracle. На відміну від традиційного макета диска бази даних, сторінки служби зберігання не повинні вирівнюватися на диску, а отже, сторінки не повинні мати однакового розміру. Сторінки мають не тільки гнучкий розмір, але навіть дозволяють збільшувати розмір сторінки. Отже, оскільки довгі записи з часом зростають незначно, записи ніколи не потрібно переміщувати.

У сценаріях, коли записи істотно зростають з часом, не можна уникнути переміщення записів. Розбиття сторінки даних має багато спільного із розділенням сторінок для В-дерева на стороні клієнта. Таким чином, може бути використана індексована впорядкована таблиця, де сторінки аркушів є сторінками даних, і всі алгоритми можуть бути безпосередньо адаптовані. Однак в індексованих впорядкованих таблицях більше не можна використовувати службу індексування для первинного індексу. Для решти дисертації вважаю, що записи не зростають суттєво, оскільки це дозволяє використовувати ту саму презентацію для індексу В-Tree, а також служби індексації. Однак усі алгоритми також працюють, з невеликими коригуваннями, з індексованою упорядкованою таблицею.

Читайте(ключ як uuid, uri як рядок): зчитує дані корисного навантаження запису з урахуванням ключа запису та URI колекції.

Оновлення(ключ як uuid, корисне навантаження як двійкове, uri як рядок): Оновлює інформацію про корисне навантаження запису. У цьому дослідженні всі ключі незмінні. Єдиний спосіб змінити ключ запису - це видалити та повторно створити запис.

Видалити(ключ як uuid, uri як рядок): видаляє запис. Сканувати (uri як рядок): сканує всі записи колекції. Для підтримки сканування диспетчер записів повертає ітератор до програми.

На додаток до методів створення, читання, оновлення та сканування, API диспетчера записів підтримує методи фіксації та скасування. Ці два методи реалізовані менеджером сторінок, як описано в наступному розділі. Крім того, диспетчер записів виставляє інтерфейс для зондування індексів (наприклад, запитів діапазону): Такі запити обробляються службами, такими як SimpleDB (просто реалізується), або індексами B-Tree, які також реалізовані поверх хмарної інфраструктури (розділ 2.4.4).

2.4.3 Менеджер сторінок

Менеджер сторінок реалізує пул буферів безпосередньо поверх API Cloud Storage. Він підтримує читання сторінок із служби, закріплення сторінок у пулі буферів, оновлення сторінок у пулі буферів та позначення сторінок як оновлених. Менеджер сторінок також надає спосіб створення нових сторінок. Вся ця функціональність проста і може бути реалізована так само, як і в будь-якій іншій системі баз даних. Крім того, менеджер сторінок реалізує методи фіксації та переривання. Використовую термін транзакція для послідовності читання, оновлення та створення запитів між двома дзвінками фіксації або скасування. Передбачається, що набір записів транзакції (тобто набір оновлених та новостворених сторінок) вписується в основну пам'ять або вторинну пам'ять клієнта (наприклад, флеш-пам'ять або диск). Якщо заявка бере участь, всі оновлення поширюються в хмару за допомогою методу put хмарної служби зберігання даних (Розділ 2.3.1), а всі постраждалі сторінки в пулі буферів позначаються як незмінені. Якщо програма скасовує транзакцію, усі сторінки, позначені як змінені або нові, просто видаляються з пулу буферів, без взаємодії з хмарною службою. У цій роботі я широко використовую термін транзакція: не всі протоколи, представлені в цій дисертації, дають гарантії ACID у сенсі БД. Припущення, що набір записів транзакції повинен вміщуватися в буферний пул клієнта, можна послабити, виділивши для цього додаткові сторінки переповнення за допомогою служби хмарного сховища; обговорення таких протоколів, однак, виходить за рамки цієї тези. Якщо програма припиняє транзакцію, усі сторінки, позначені як змінені або нові, просто видаляються з

пулу буферів, без взаємодії з хмарною службою. У цій роботі я широко використовую термін транзакція: не всі протоколи, представлені в цій дисертації, дають гарантії ACID у сенсі БД. Припущення, що набір записів транзакції повинен вміщуватися в буферний пул клієнта, можна послабити, виділивши для цього додаткові сторінки переповнення за допомогою служби хмарного сховища; обговорення таких протоколів, однак, виходить за рамки цієї тези. Якщо програма скасовує транзакцію, усі сторінки, позначені як змінені або нові, просто видаляються з пулу буферів, без взаємодії з хмарною службою. У цій роботі я широко використовую термін транзакція: не всі протоколи, представлені в цій дисертації, дають гарантії ACID у сенсі БД. Припущення, що набір записів транзакції повинен вміщуватися в буферний пул клієнта, можна послабити, виділивши для цього додаткові сторінки переповнення за допомогою служби хмарного сховища; обговорення таких протоколів, однак, виходить за рамки цієї тези. Не всі протоколи, представлені в цій дипломній роботі, дають гарантії ACID у сенсі БД. Припущення, що набір записів транзакції повинен вміщуватися в буферний пул клієнта, можна послабити, виділивши для цього додаткові сторінки переповнення за допомогою служби хмарного сховища; обговорення таких протоколів, однак, виходить за рамки цієї тези. Не всі протоколи, представлені в цій дипломній роботі, дають гарантії ACID у сенсі БД. Припущення, що набір записів транзакції повинен вміщуватися в буферний пул клієнта, можна послабити, виділивши для цього додаткові сторінки переповнення за допомогою служби хмарного сховища; обговорення таких протоколів, однак, виходить за рамки цієї тези.

Менеджер сторінок зберігає копії сторінок із служби хмарного зберігання в буферному пулі між транзакціями. Тобто, жодна сторінка не виселяється з пулу буферів як частина коміту. Аборт виселяє лише змінені та нові сторінки. Сторінки оновлюються в пулі буферів із використанням протоколу TTL: якщо до буферного пулу буде запропоновано немодифіковану сторінку після закінчення терміну її роботи, менеджер сторінок видає get-ifmodified-, оскільки запит на хмарне сховище API для отримання оновленої версії, якщо це необхідно (Розділ 2.3.1). Крім того, менеджер сторінок підтримує примусове отримання get-

ifmodified-, оскільки запити під час отримання сторінки корисні для перевірки сторінок.

2.4.4 Інденси

Як уже зазначалося, існує два принципово різні способи реалізації індесів. По-перше, можна використовувати хмарні служби для індесації, такі як SimpleDB. По-друге, інденси можуть бути реалізовані поверх менеджера сторінок. Цей розділ описує способи реалізації обох підходів.

Індекс B-Tree

B-Trees можуть бути реалізовані поверх менеджера сторінок досить просто. Знову ж таки, ідея полягає в тому, щоб якомога більше застосувати існуючу технологію баз даних підручників. Кореневий та проміжний вузли B-Tree зберігаються як сторінки в службі зберігання (через менеджер сторінок) і містять пари (ключ, uri): uri посилається на відповідну сторінку на наступному нижчому рівні. Листові сторінки первинного індесу містять записи форми (ключ, PageURI) і, таким чином, посилаються на сторінки відповідних записів (Розділ 2.4.2). Сторінки вторинного індесу містять записи форми (пошук ключ, ключ запису). Отже, зондування вторинного індесу передбачає навігацію по вторинному індесу з метою отримання ключів відповідних записів, а потім навігацію по первинному індесу з метою отримання записів з даними корисного навантаження.

Як згадано у Розділі 2.4.1, блокування блокувань слід уникати якомога довше в масштабованій розподіленій архітектурі. Тому я пропоную використовувати дерева B-link[25] та їх використання в розподіленій системі для того, щоб дозволити одночасне читання та запис (зокрема, розділення), а не більш загальноприйнятий протокол з'єднання блокування. Тобто кожен вузол B-дерева містить покажчик (тобто URI) на своєму правому браті на тому ж рівні. На рівні листів цей ланцюжок можна природно використовувати для здійснення сканування у всій колекції або через великі діапазони ключів.

Кожне B-дерево ідентифікується за URI інформації про його індекс, що містить URI кореневої сторінки індесу. Це гарантує, що коренева сторінка

завжди доступна через один і той же URI, навіть коли коренева сторінка розділяється. Колекція в нашій реалізації ідентифікується за URI інформації про її первинний індекс. Усі URI до інформації індексу (первинної чи вторинної) постійно зберігаються як метадані в каталозі системи на хмарній службі (Розділ 2.4.1).

Індекс хмарності

Використання хмарного API, визначеного в розділі 2.3.7, реалізація вторинного індексу є простою. Записи такі ж, як для реалізації B-Tree, і мають форму (ключ пошуку, ключ запису). Однак, оскільки API визначає ключ пошуку, а також ключ запису як рядок, числові значення повинні кодуватися. Можливості кодування числових клавіш пошуку. Оскільки індекс надається як послуга, інформація про індекс не потрібна для пошуку кореневої сторінки дерева. Однак для первинних індексів інформація про індекс все ще потрібна для зберігання таблиці вільного простору. За винятком цього, записи мають ту саму форму, що і для B-Tree, і можуть бути реалізовані таким же чином, як вторинні індекси.

2.4.5 Протоколювання

Протоколи, описані в Розділі 3.5, широко використовують записи повторних журналів. У всіх цих протоколах передбачається, що записи журналу є ідемпотентними; тобто застосування журналу двічі або частіше має такий самий ефект, як застосування журналу лише один раз. Знову ж таки, немає необхідності заново винаходити записи про колеса та журнали підручників, а також методи ведення журналу є доречними. Якщо не вказано інше, використовуйте наступні (прості) записи журналу переробки в нашій реалізації:

(вставка, ключ, корисне навантаження): запис журналу вставки описує створення нового запису; такий запис журналу завжди пов'язаний з колекцією (точніше з первинним індексом, який організовує колекцію), або з вторинним індексом. Якщо такий запис журналу вставки пов'язаний з колекцією, тоді ключ представляє ключове значення нового запису, а корисне навантаження містить інші дані запису. Якщо запис журналу вставки пов'язаний із вторинним індексом, тоді ключовим є значення ключа пошуку цього вторинного індексу (можливо,

складеного), а корисне навантаження - значення первинного ключа референтного запису.

(видалити, ключ): запис журналу видалення також пов'язаний або з колекцією (тобто первинним індексом), або з вторинним індексом.

(update, key, afterimage): запис журналу оновлення повинен бути пов'язаний зі сторінкою даних; тобто листовий вузол первинного індексу колекції. Запис журналу оновлення містить новий стан (тобто після зображення) згаданого запису.

За своєю природою всі ці записи журналу є ідемпотентними: у всіх трьох випадках з бази даних можна визначити, чи оновлення, описані записом журналу, вже застосовані. Однак із такими простими записами журналу оновлення можливо одне і те ж оновлення застосовуватись двічі, якщо інше оновлення замінило перше оновлення до другого оновлення. Щоб уникнути цих невизначень, можна використовувати більш складні журнали, такі як журнальні записи, що використовуються в Розділі 2.5.2.

Якщо операція передбачає оновлення запису та оновлення одного або декількох вторинних індексів, тоді диспетчер записів створює окремі записи журналу для реєстрації оновлень у колекції та вторинних індексах. Знову ж таки, реалізація цієї функції в диспетчері записів є простою і не відрізняється від будь-якої системи баз даних підручників.

Більшість протоколів, що вивчаються в цій роботі, включають лише повторне реєстрацію. Тільки протокол, наведений у Розділі 2.5.4, вимагає скасування реєстрації. Скасувати ведення журналу також нескладно реалізувати, зберігаючи попереднє зображення на додаток до останнього зображення в записах журналу оновлення та зберігаючи останню версію запису в записах журналу видалення.

Висновки до розділу

У відкритій архітектурі, зображеній на рисунку 2.1, існує декілька проблем безпеки. Найважливіші з них - це контроль доступу до даних та питання довіри до хмарного провайдера. Далі я коротко опишу можливі шляхи вирішення обох проблем безпеки. Однак більш детально висвітлити аспект безпеки виходить за рамки цієї дипломної роботи і залишається подальшою роботою.

Базовий контроль доступу може бути реалізований за допомогою функціональності контролю доступу, що використовується в більшості хмарних служб. Наприклад, усередині Amazon S3 користувачі можуть надавати іншим користувачам права на читання та / або запис на рівні сегментів або окремих об'єктів усередині сегмента. Цей грубозернистий контроль доступу може бути використаний для основного захисту даних. Більш детальна безпека та гнучка авторизація, наприклад, подання SQL, повинні бути реалізовані поверх постачальника сховища. Залежно від довіри до користувача та стеку додатків застосовується кілька сценаріїв безпеки для реалізації більш детального контролю. Якщо протоколи неможливо порушити, одним із рішень є призначення куратора для кожної колекції. У цьому випадку всі оновлення повинні бути схвалені куратором, перш ніж вони стануть видимими для інших клієнтів. Поки оновлення чекають на затвердження, клієнти продовжують бачити стару версію даних. Це забезпечує більш центральний спосіб контролю безпеки і добре підходить для протоколу контрольної точки, представленого в наступному розділі. Крім того, куратор може забезпечити обмеження цілісності всередині бази даних. Однак куратор працює лише для простих протоколів, які не надають гарантій серіалізації. Для вдосконалених протоколів або випадків їх пошкодження література P2P вже пропонує різні рішення куратор працює лише для простих протоколів, які не надають гарантій серіалізації. Для вдосконалених протоколів або випадків їх пошкодження література P2P вже пропонує різні рішення куратор працює лише для простих протоколів, які не надають гарантій серіалізації. Для вдосконалених протоколів або у випадку їх пошкодження,

література P2P вже пропонує різні рішення[3], які також адаптовані до представленої тут архітектури.

Проблему довіри між користувачем або програмою та постачальником послуг можна вирішити зашифрувавши всі дані. Крім того, щоб надати різним наборам клієнтів доступ до різних наборів сторінок, можна використовувати різні ключі шифрування. У цьому випадку заголовок сторінки вказує, який ключ потрібно використовувати для дешифрування та повторного шифрування сторінки у разі оновлень, і цим ключем повинні користуватися всі клієнти або сервери додатків, які можуть отримати доступ до цієї сторінки.

РОЗДІЛ 3

РОЗРОБКА ОНЛАЙНОВОГО РЕПОЗИТОРІЮ ПРОЕКТІВ КОМПАНІЇ ДЛЯ ООНОВЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Описання принципів роботи розробленої системи контролю версій в в онлайнному репозиторії проектів

Дані, що використовуються для тестів, беруться з тесту TPC-W. Тест TPC-W імітує інтернет-книгарню та імітує перегляд та покупки користувачів на своєму веб-сайті. Таблиці даних вказують усі важливі сутності, які використовуються. Рисунок 3.1 показує таблиці графічно. Для мікрОВизначення в наступному розділі я використовую лише підмножину всіх таблиць.

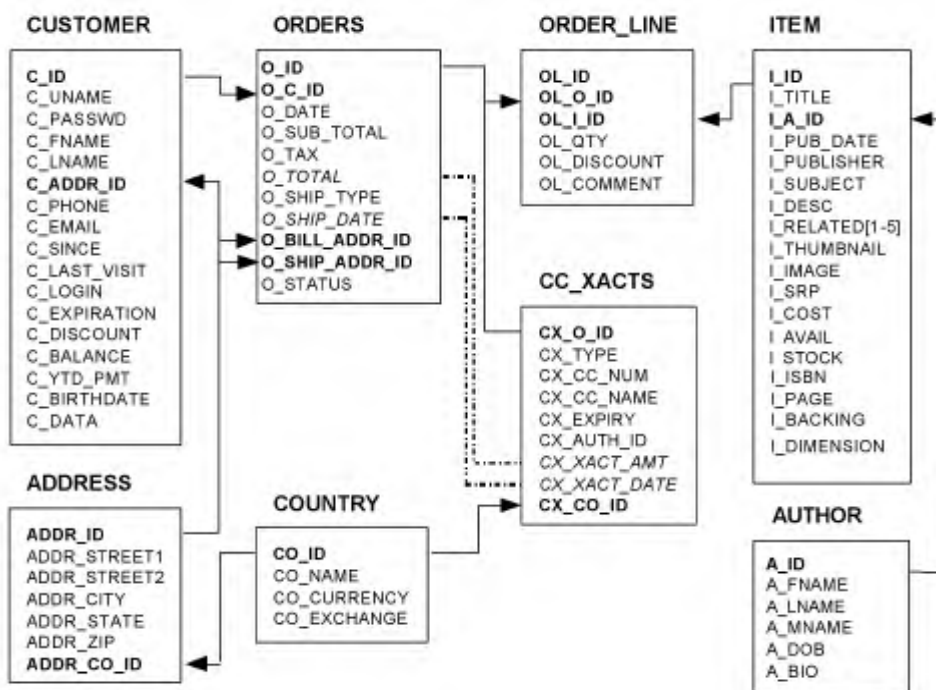


Рисунок 3.1. Таблиці даних еталону TPC-W

Кафедра КСУ				НАУ 21 08 21 000 ПЗ			
Виконав	Орнатська С.Д.			Розробка онлайнного репозиторію проектів компанії для оновлення програмного забезпечення	Літера	Аркуш	Аркушів
Керівник	Росінська Г.П.				Д	64	72
Консульт.					СП 501Бз 123		
Норм. контр.	Тупота С.В.						
Зав. Каф.	Литвиненко О.Є.						

3.2 Мікробенчмаркінг

This section presents the microbenchmark used to compare the performance of different setups of the system. The microbenchmark tests the system with different simple queries which cover all access paths and also insert and delete. The systems are tested under increasing load by concurrently issuing the queries.

3.2.1 Налаштування системи

Тестуються та порівнюються різні установки:

Налаштування MySQL InnoDB: Один примірник сервера MySQL, що використовує InnoDB як рівень зберігання.

Налаштування S3 One: Один вузол S3 із запущеним екземпляром MySQL, що використовує S3 як рівень зберігання.

Налаштування S3 Five Replica: П'ять вузлів S3, кожен з яких запускає екземпляр MySQL

використання S3 як рівня зберігання з коефіцієнтом реплікації даних "п'ять". Це означає, що як тільки дані записуються в систему, вони реплікуються на всі вузли. Налаштування зі стандартним MySQL із механізмом зберігання InnoDB використовується для порівняння власної реалізації рівня зберігання та стандартного MySQL. У цьому випадку я використовую InnoDB, оскільки це один із найбільш широко використовуваних механізмів зберігання. Різні установки S3 використовуються для того, щоб показати, чи кількість вузлів у S3 впливає на продуктивність систем.

Апаратне забезпечення, що використовується для проведення тестів, - це п'ять 64-розрядних машин з двоядерним процесором AMD Opteron 280 та 4 ГБ оперативної пам'яті. На кожному працює Windows 7. Всі машини працюють в одній внутрішній мережі.

3.2.2 Запити

У цьому мікровизначенні використовується ряд простих запитів. Я намагався охопити всі шляхи доступу, а також включив набір запитів для перевірки вставки та видалення продуктивності системи. Ось список усіх запитів:

Запит первинного ключа

Наступний запит видається 2000 разів поспіль за пробіг, щоб отримати номер продуктивності:

ВИБЕРІТЬ * 3 елемента ДЕ i_id = <випадкове число>

Таблиця товарів містить 10000 позицій. Випадкове число є дійсним ідентифікатором одного з елементів у списку. i_id attribute є первинним ключем для таблиці, і цей запит в основному перевіряє шлях доступу для доступу до первинного ключа.

Запит вторинного індексу

Наступний запит виконується 250 разів поспіль за пробіг:

ВИБРАТИ кількість (i_id) ВІД елемента ДЕ i_subject =

"<випадкова тема>"

Для атрибута створюється вторинний індекс i_subject. Таблиця товарів містить 10000 найменувань, а є 24 різніпредметів в системі. Кожен предмет повинен посилатись приблизно на 400 предметів. Функція підрахунку в запиті забезпечує, що кожен елемент теми повинен отримати доступ. За допомогою цього запиту я перевіряю шлях доступу за вторинними індексами.

Повне сканування таблиці

Наступний запит виконується 50 разів у циклі за один цикл:

ВИБРАТИ кол (ol_id) ВІД рядка_замовлень

Існує 7750 записів у рядок_замовлень таблиця. За допомогою цього запиту до кожного запису потрібно отримати доступ і видається повне сканування таблиці.

Запит бестселера

Так званий запит бестселера використовується для тестування більш складного запиту, який отримує доступ до багатьох таблиць. Запит бестселера отримує 50 найпопулярніших товарів за кількістю замовлень із останніх 3333

замовлень. Я не показую тут запит і вказую зацікавленим читачам на специфікацію TPC-W. У тестах бестселер запускається лише один раз за серію.

Вставлення та видалення запиту

Для тестування вставки та видалення продуктивності бази даних я вставляю і згодом видаляю ті самі десять записів у автор таблиця з наступними запитом. Таблиця автора також містить вторинний індекс на `lname` атрибут. Отже, обидва шляхи вставки покриті. Запити на вставку та видалення виконуються 1000 разів за пробіг.

3.2.3 Результати

Бенчмарк був запущений з різними установками, як описано вище на згаданому обладнанні. Усі запити, продемонстровані в попередньому розділі, були запущені. Кожна точка вимірювання - це загальний середній час 25 послідовних прогонів. Результати детально показані в таблицях 3.1, 3.2 та 3.3. Всі виміряні рази даються секундами. У тому випадку, коли працювало більше одного екземпляра MySQL, як при установці з п'ятьма вузлами S3, підключення до екземплярів MySQL були рівномірно розподілені між усіма запущеними екземплярами.

На жаль, деякі показники для налаштування S3 із запитом вставки та видалення не вдалося виміряти до кінця. Відповідні поля залишились порожніми та позначені *. Запустивши вставку та видалення еталону під цим носієм завантаження вже починаючи з 25 одночасних потоків, налаштування S3 ставали повільнішими та повільнішими. Тести були скасовані, коли один пробіг, як виявилось, зайняв більше десяти хвилин. Одним з побічних ефектів, який можна було відстежувати, було те, що споживання пам'яті ставало все більшим і більшим, доки не досягло максимальної пам'яті кучі, призначеної для S3, майже чотирьох гігабайт пам'ятної пам'яті. Коли це число було досягнуто, система почала смітити, а вимірювані часи почали зростати в геометричній прогресії. У деяких випадках база даних виходила з ладу, а разом з нею навіть серверна машина. Однак пам'ять, необхідна для зберігання цілих даних, порівняно мала лише кілька сотень мегабайт.

Таблиця 3.2 показує, що в установці лише з одним вузлом S3 поріг продуктивності зчитування вже досягнуто при приблизно 100 одночасних зверненнях. InnoDB все ще працює досить добре з 200 потоками, навіть запити на вставку та видалення успішно закінчуються під цим високим навантаженням.

3.2.4 Аналіз масштабування S3 з різною кількістю вузлів

Табле 3.4 дає порівняння часу роботи між установкою з одним вузлом S3 та установкою з п'ятьма вузлами S3. Оскільки цифри вказують, що система, здається, масштабується і може витримати набагато більше навантаження, якщо кількість вузлів S3 більша. Це особливо видно у випадку, коли 100 потоків одночасно отримують доступ до системи. Налаштування з одним вузлом S3, здається, досягає порогового значення, тоді як установка з п'ятьма вузлами все ще працює добре, і це вказує на колосальну різницю в 2694% часу роботи.

Порівняння S3 з InnoDB

Таблиця 3.5 представляє продуктивність S3 порівняно зі стандартною установкою MySQL з InnoDB. Майже лінійне збільшення продуктивності бази даних S3 порівняно з InnoDB при зростаючому навантаженні до 100 потоків виглядає перспективним. Однак з останнім вимірюванням з використанням 200 та 300 потоків продуктивність системи S3 порівняно з InnoDB вже починає знову знижуватися.

Крім того, цей мікробенчмарк дозволяє лише порівнювати ефективність з рівними запити. Вузьке місце для бази даних S3 полягає особливо в продуктивності вставки та видалення. Як пояснювалося вище, час роботи просто вибухає, коли одночасно видається занадто багато запитів на оновлення. У змішаних умовах, коли видаються всілякі запити, що є звичайним випадком для бази даних у веб-середовищі, ці запити на оновлення просто знижують продуктивність усієї системи.

Таблиця 3.1. Результати для MySQL з InnoDB сторажний двигун.

Кількість ниток	1	10	25	50	100	200	300
ключовий запит	0,658	0,687	1.263	2,971	6.227	35,982	62.230

вторинний індекс	0,235	0,558	1.474	2.998	6.642	12.750	19.294
повне сканування таблиці	0,108	0,315	1.296	3.395	6.643	13.814	21.294
бестселер	0,097	0,466	0,998	1,888	4.582	7,669	12.901
вставити та видалити	1.212	2,968	7.215	17.343	249,687	413,176	501,816

3.3. Висновки до розділу

В третьому розділі розкрито етапи розробки системи контролю доступу і управління системою контролю версій програмного забезпечення через корпоративну мережу (на прикладі банківської комп'ютерної мережі). Було описано мову програмування, яка використовувалась для розробки системи. Визначено необхідне апаратне забезпечення. Описано реалізацію алгоритмів системи. Розкрито функції модуля адміністрування та представлено загальний інтерфейс системи.

Також у цьому розділі реалізовано перехід від лінійки продуктів до програмної екосистеми.

Після відбору вибірки відповідного типу екосистеми програмного забезпечення було згадано питання трансформації лінійки продуктів у відкриту платформу, взаємних відносин інженерів-розробників та наслідків для розробки програмного забезпечення.

ВИСНОВКИ

В дипломному проєкті в повній мірі виконано поставлену задачу – розроблено репозиторій проєктів компанії з можливістю контролю версій *web*-ресурсів, за раунок зменшення часу на пошук і розгортання програмного забезпечення.

Було визначено три типи програмних екосистем:

- на основі операційних систем;
- на основі прикладних програм;
- на основі кінцевого користувача.

Веб-додатки потребують високої масштабованості та доступності за низьких та передбачуваних витрат. Жоден користувач ніколи не повинен бути заблокований ні іншими користувачами, що отримують доступ до тих самих даних, ні через несправності обладнання постачальника послуг. Натомість користувачі очікують постійного та передбачуваного часу відгуку під час взаємодії з веб-службою. Службові обчислення (так звані хмарні обчислення) можуть потенційно задовольнити всі ці вимоги. Хмарні обчислення спочатку були розроблені для конкретних навантажень. У цій главі показані можливості та обмеження застосування хмарних обчислень до робочих навантажень загального призначення, використовуючи для прикладу AWS та, зокрема, S3. У цьому розділі показано, як архітектуру підручника для побудови систем баз даних можна застосувати для побудови системи хмарних баз даних. Крім того, у цій главі було представлено кілька альтернативних протоколів узгодженості, які зберігають філософію дизайну хмарних обчислень та торгових витрат та доступності для вищого рівня узгодженості. Нарешті, важливим внеском цього розділу було вивчення альтернативних архітектур клієнт-сервер та індексація для здійснення додатків та пошуку індексів.

Це дослідження є важливим для повного розуміння того, що таке екосистеми та як ними можна керувати. Різні підходи та результуюча точка показують наслідки залежності в різних випадках взаємодії, але головна ідея фактора високого коефіцієнта успіху полягає в тому, що сильний взаємозв'язок

та наявність усіх необхідних компонентів екосистем забезпечують необхідний результат при мінімальних витратах часу. Дослідження, якщо воно визначене, є основою для всебічного моделювання кожної системи, яка зацікавлена в обсязі домену. За допомогою простих інструментів та повної графічної моделі легко спостерігати за відсутніми посиланнями та акторами з їхніми послідовними елементами, а у разі потреби – створювати нові підсистеми, рекламувати нові компоненти та спостерігати за результатами.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Backlund Alexander "The definition of system". In: *Kybernetes* Vol. 29 nr. 4, – 2000 – pp. 444–451.
2. *Biology Concepts & Connections Sixth Edition*”, Campbell, Neil A. (2009), page 2, 3 and G-9. Retrieved 2010-06-14
3. *From Integration to Composition: On the Impact of Software Product Lines, Global Development and Ecosystems*, Accepted for *Journal of Systems and Software*, June 2009. Jan Bosch and Petra Bosch-Sijtsema Intuit Inc, Mountain View, CA, USA
4. *From Software Product Lines to Software Ecosystems* Jan Bosch Intuit, 2500 Garcia Avenue, Mountain View, CA 94043. Accepted for the 13th International Software Product Line Conference (SPLC 2009) August 24-28, 2009, San Francisco, CA .
5. Graves S., Dorai-Raj S. *Creating R Packages, Using CRAN, R-Forge, And Local R Archive Networks And Subversion (SVN) Repositories*. 2018.
6. Graves S., Dorai-Raj S. *Creating R Packages, Using CRAN, R-Forge, And Local R Archive Networks And Subversion (SVN) Repositories*. 2008. Толковый словарь компьютерных технологий: Пер. с англ./Митчелл Шниер – К.: Издательство “ДиаСофт”, 2010. – 720 с.
7. Marill J. L., Luczak E. C. *Evaluation of digital repository software at the national library of medicine // D-Lib Magazine*. 2017. Т.15. № 5/6. С. 1082-9873.
8. Marill J. L., Luczak E. C. *Evaluation of digital repository software at the national library of medicine // D-Lib Magazine*. 2009. Т.15. № 5/6. С. 1082-9873.
9. Mens, Tom; Demeyer, Serge (Eds.): *Software Evolution*. –Springer-Verlag Berlin Heideberg – 2008, 347 p.
10. Messersmitt D.G., Szyperski C. *Software Ecosystems: Understanding an Indispensable Technology and Industry*. – MIT press. – 2003. – 233 p.
11. Onions, Charles, T. (ed) (1964). *The Shorter Oxford English Dictionary*. Oxford: Clarendon Press. p. 2095.
12. P. Dini, N. Rathbone, M. Vidal, P. Hernandez, P. Ferronato, G. Briscoe, S. Hendryx: *The digital ecosystems research vision: 2010 and beyond*

13. *Sidorov N.A., Software Ecology*, 15.02.2010
14. *Tanenbaum Andrew S Computer Networks., Prentice Hall*, 2016.
15. Автоматизация расчетных операций и фондовых бирж. – М.: Церих, 2012. – 206 с.
16. Автоматизированные информационные технологии в экономике: Учебник /Под ред. Г.А. Титоренко. – М.: Компьютер, ЮНИТИ, 2008. – 400 с.
17. Буравцев А.В. Складні технологічні системи
18. Гарсиа-Молина, Гектор, Ульман, Джеффи, Д., Уидом, Джениффер. Системы баз данных. Полный курс.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 1088 с.
19. Дайітбегов Д. М. Інтернет-репозиторій освітніх ресурсів ХТРЕІУ як інструментарій електронного навчання
20. Довгалюк Д.О. Онлайн система контролю версій *web*-ресурсів// Тези доповідей наук.-практ. конф. “Сучасні тенденції розвитку системного програмування” (25-26 листопада 2020 р.). – К.: НАУ, 2020. – С. 35.
21. Ирвин Дж., Харль Д. Передача данных в сетях: инженерный подход: Пер. с англ. СПб.: БХВ-Петербург., 2003. — 448 с.
22. Копайгородський А. Н., Массел Л. В. Розробка та інтеграція основних компонентів інформаційної інфраструктури наукових досліджень
23. Монахов С.В., Савіних В.П., Цветков В.Я. Методологія аналізу та проектування складних інформаційних систем.
24. Толковый словарь по вычислительной технике; Пер. с англ. – М.: ТОО «*Channel Trading Ltd.*», 2015.
25. НД ТЗІ 1.1-003-99. Термінологія у області захисту інформації в комп'ютерних системах від несанкціонованого доступу. // Департамент спеціальних телекомунікаційних систем і захисту інформації Служби безпеки України. – Київ, 1999.
26. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

Додаток А

Лістинг коду основного програмного модулю