Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр _____післядипломної освіти_____
(повна назва)

Кафедра _____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

Дослідження методів впровадження технологічних стеків при розробці веб-додатків

(тема)

Виконав: студент 2 курсу, групи ІПЗмзд-18-1__

спеціальності _121 – Інженерія програмного забезпечення_____
(код і повна назва спеціальності)

освітньо-наукової програми _Інженерія програмного_забезпечення_ _____
(повна назва освітньої програми)

_____Радіонова А.М._____
(прізвище, ініціали)

Керівник _____проф. Шубін І.Ю._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти

Кафедра програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

Освітньо-наукова програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____ 20 __ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

Студентові *Радіоновій Анастасії Миколаївні*
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів впровадження технологічних стеків при розробці веб-додатків

затверджена наказом по університету від «___» _____2020 р № ____

2. Термін подання студентом роботи до екзаменаційної комісії «18» травня 2020 р.

3. Вихідні дані до роботи*: веб-орієнтована технологія технологічних стеків, пояснювальна записка. Використовувати ОС Windows, середовище об'єктно-орієнтованого проектування.*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, методи пошуку корисних даних, опис об'єктних моделей, використовувані. методи та алгоритми, архітектура програмної системи, опис розробленої програмної системи. , результати тестування програмної системи*

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта	
		підпис	дат
Спецчастина	проф. Шубін І.Ю.		

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	25 березня 2020 р.	
2.	Огляд існуючих методів	31 березня 2020 р.	
3.	Методи застосування програмних стеків	15 квітня 2020 р.	
4.	Підготовка пояснювальної записки	20 квітня 2020 р.	
5.	Спецчастина	28 квітня 2020 р.	
6.	Підготовка презентації та доповіді	10 травня 2020 р.	
7.	Попередній захист	12 травня 2020 р.	
8.	Нормоконтроль, рецензування	13 травня 2020 р.	
9.	Занесення диплома в електронний архів	15 травня 2020 р.	
10	Допуск до захисту в зав. кафедри	18 травня 2020 р.	

Дата видачі завдання _ «____» _____2020 р.

Студент _____
(підпис)

Керівник _____ проф. Шубін І.Ю..
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи : 90 с., 40 рис., 6 табл., 3 дод., 27 джерел.

ТЕХНОЛОГІЧНИЙ СТЕК, ФРОНТ-ЕНД, АЛГОРИТМИ, ХМАРНІ СХОВИЩА, ІНТЕГРАЦІЯ, СИНХРОНІЗАЦІЯ, ДАНІ, СТРУКТУРИ

Об’єктом дослідження є технології розробки веб-застосувань.

Методом дослідження є комп’ютерне моделювання в спеціалізованих пакетах програм.

Метою роботи є аналіз і розробка алгоритмів покращення роботи зі технологічним стеком та інтеграції даних у хмарних сховищах.

Результатом атестаційної роботи магістра є алгоритм інтеграції даних у хмарних сховищах на основі використання технологічного стеку при проектуванні та було розроблено програмний продукт, який моделює виповнення алгоритмів інтеграції даних у хмарних сховищах.

ALGORITHMS, FRONT-END, BACK-END, CLOUD STORAGE, INTEGRATION, SYNCHRONIZATION, DATA, STRUCTURE

Work is devoted to a cloud storage data synchronization problem.

The method of research is a specialized computer modeling software packages.

The aim is to analyze and develop algorithms of data integration in cloud storages.

The result of attestation master's degree work is the algorithm for data integration in cloud storages - developed software that simulates execution of algorithm in data integration in cloud storages.

ЗМІСТ

ВСТУП

Термін «Стек технологій» згадується зазвичай, коли говорять про портфель технологій, з якими працює та або інша компанія, або коли потрібно сказати/запитати про багаж знань, яким має людей. Стек або технології стеків – термін, який ставиться до набору технологій, програмного забезпечення й інструментів, які використовуються в розробці й розгортанні сайтів, додатків або інших цифрових продуктів.

У веб-розробці три загальні набори стеків – Front-end, Back-end і Full Stack – використовуються для опису ролі розроблювачів програмного забезпечення, оскільки вони творці кінцевого продукту. Усе більше й більше професіоналів воліють працювати як з Front-end так і з Back-end одночасно, щоб збільшити свою гнучкість і стати Full-stack розроблювачем.

Існують дві очевидні причини користування «Хмарами»: синхронізація файлів між декількома обладнаннями й створення резервних копій файлів.

Основна перевага «хмар» – це можливість доступу до файлів з будь-якого місця, де є підключення до мережі інтернет. Зараз більшість компаній, які надають послуги хмарних сховищ, надають певна кількість місця у використання безкоштовно. Додаткове місце в «хмарі» надається за окрему плату. Дуже часто людей користується не одним, а декількома хмарними сховищами. У такому випадку, згодом може настати момент, коли людині знадобитися синхронізувати дані на різних сховищах, або перенести файли з один на інше.

Однак, навіть із обліком кешу, є певний програш у продуктивності, якщо врахувати час на парсінг і виконання скрипту. Тільки jquery може гальмувати деякі мобільні браузери на сотні мілісекунд.

Що ще гірше, зазвичай користувач не одержує ніякого фідбеку в той час, як завантажуються скрипти. Результат — чиста сторінка на екрані, яка потім раптово перетворюється в повністю завантажену сторінку.

Більші скрипти завантажуються набагато довше, чим видасться. Потрібно

«чотири обміни пакетами й сотні мілісекунд затримки, щоб вийти на 64 КБ обміну даними між клієнтом і сервером». Оскільки це правило діє також для первісного завантаження сторінки, те дуже важливо, який контент вантажиться для рендерингу на сторінці в першу чергу.

Веб-сайти, яким вдається доставити контент (нехай навіть базову розмітку без даних) у цьому вікні, видадуться винятково чуйними. Насправді, багато авторів швидких серверних додатків сприймають Javascript як щось непотрібне або що потрібно використовувати з великою обережністю. Таке відношення ще більше підсилюється, якщо в додатка швидкі бекенд і база даних, а його сервери перебувають біля користувачів.

Роль сервера в прискоренні представлення контенту прямо залежить від веб-додатку – рішення не завжди зводиться до «рендерінгу цілих сторінок на сервері». Найбільший недолік продуктивності в багатьох популярних системах у наш час пояснюється прогресивним нагромадженням складності в стеці. Згодом додавалися технології начебто Javascript і CSS. Їхня популярність теж поступово росла. Тільки зараз ми можемо оцінити, як їх можна використовувати по-іншому. Мова йде й про поліпшення протоколів (це показує нинішній прогрес SPDY і QUIC), але найбільшу вигоду несе все-таки оптимізація додатків.

Актуальність даної роботи полягає в необхідності ретельного аналізу «хмарних» програмних продуктів з метою об'єктивного розуміння таких основних факторів як їхня конкурентоспроможність, принцип дії, сфера застосування й можливі тенденції розвитку. Важливим також є дослідження проблем, протиріч і можливих обмежень «хмарного» ПЗ, оскільки необхідно, у цьому випадку, максимально знизити ризик втрати інформації й забезпечити найбільшу продуктивність при роботі з даними в хмарі.

Потенціал даного сегменту сервісів дуже великий і є велика кількість людей і компаній, що активно прибігає до його використання. Саме тому важливо одержати докладне представлення про новітні технології, не тільки будь-якому фахівцеві, який зв'язує свою поточну або майбутню діяльність із сучасними інформаційними технологіями, але й рядовому користувачеві.

1 ОГЛЯД ДЖЕРЕЛ, АНАЛІЗ ВИРІШЕННЯ ПРОБЛЕМ, ОБҐРУНТУВАННЯ ЦІЛЕЙ ДОСЛІДЖЕННЯ

1.1 Основні визначення й цілі створення технологічних стеків

Даний тип розробки містить у собі всі частини веб-сайту, який користувач може бачити й взаємодіяти з ними. При розробці сайту, ціль Front-end розроблювачів переконатися, що сайт легкий для використання його користувачами які будуть заходити на сайт. За Front-end відповідають три основні мови:

- HTML – структура веб-сторінки, яка містить контент;
- CSS – стайлинг HTML елементів;
- Javascript – відповідає за всі взаємодії з користувачем

Розробка Back-end фокусується на серверах, додатках і базах даних. Розроблювачі создавают і підтримують технологію цих трьох компонентів, використовуючи безліч мов програмування, бібліотек і інших існуючих програм. Для складних веб-сайтів, які вимагають можливості за межами інтерфейсних ключів, back-end є необхідністю.

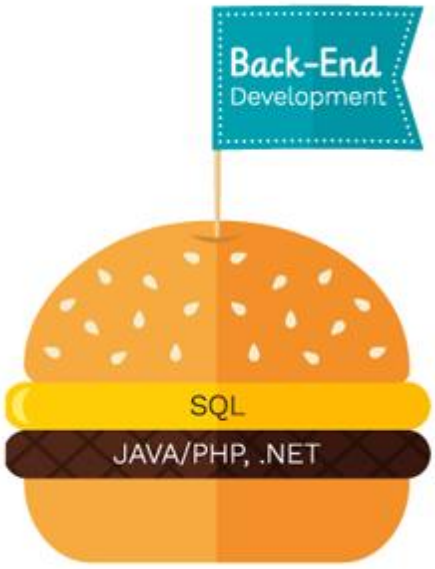

Рисунок 1.1 – Технологія «Back-end»

Back-end містить у собі розробку на таких мовах як Java, .NET а також баз даних і інших надійних платформ, таких як MySQL і Amazon AWS. По суті, прагне створити динамічний кінцевий продукт, який вимагає сервера й бази даних, щоб відправити потрібну інформацію в потрібний час.

У той час як у минулому розроблювачі були розділені між Front-end (веб-дизайн) і Back-end (розробка Web), зараз стало більш розповсюдженим для програмістів осідлати забір між двома компетенціями.

Рисунок 1.2 – Ілюстрація поняття «Full-stack»

Якщо розроблювачі є Full stack, вони комфортно можуть працювати із двома стеками відразу. Це означає, що в них є спеціальні знання всіх етапів розробки програмного забезпечення. Ці розроблювачі мають навички роботи з різними компонентами кожного типу розвитку, включаючи бази даних, HTML, Javascript, CSS і багато чого іншого.

Веб-сайти складаються з безлічі різних технологій, які часто можуть мінятися залежно від потреб а також природньої еволюції. Коли розроблювачі прагнуть поліпшити веб-сайт новими функціями або навіть зменшити

високотехнологічні компоненти, вони можуть дивитися повз стандартний розчин стеків щоб задовольнити їхні конкретні потреби.

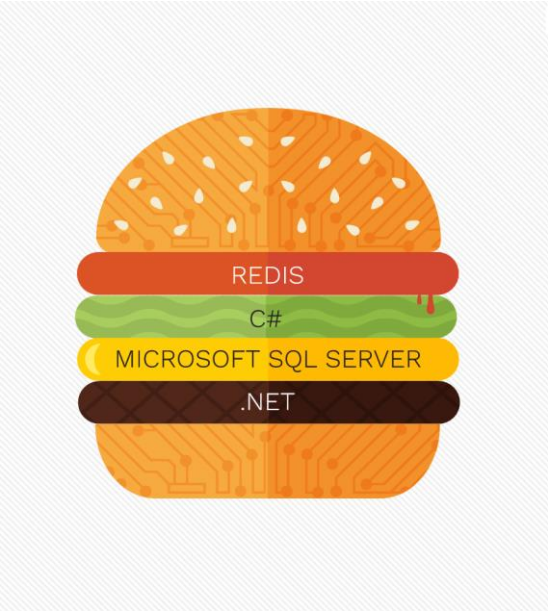

Рисунок 1.3 – Технологічні стеки

Розроблювачі можуть виготовити веб-сайти й додатка, використовуючи майже нескінченну комбінацію технологій, починаючи від мов програмування, бази даних, утиліт, бізнес-інструментів і багато чого іншого. Деякі веб-сайти можуть мати невеликий діапазон тільки від трьох до п'яти базових технологій, у той час як інші, можливо, зажадають 15 і більше. Коротше кажучи, технологія стеків залежить від цілей кінцевого продукту.

1.2 Аналіз технологи стека MEAN і Mean-програмування

Одним з перших наборів веб-технологій з відкритим вихідним кодом, що одержали широку популярність, став стек LAMP. Для створення веб-сторінок на основі HTML використовувалися операційна система Linux®, веб-сервер Apache, база даних Mysql і мова програмування Perl (або Python, або PHP). Ці технології не призначали для спільної роботи. Це окремі проекти, які зібрав воєдино один

цілеспрямований інженер-програміст — а потім інший, і ще, і ще. З тих пор ми стали свідками «кембрійського вибуху» розмноження веб-стеков. Мабуть, кожна сучасна мова програмування має своє веб-середовище (а то й дві), у якому зібраний строкатий набір технологій для швидкого й простого створення нового веб-сайту.

Останнім часом у веб-співтоваристві багато говорять про новий стеку MEAN: MongoDB, Express, AngularJS, Node.js. Стік технологій MEAN відбиває сучасний підхід до веб-розробки: коли на кожному рівні додатка, від клієнта до сервера й персистентности, застосовується той самий мова (Javascript). У цій серії статей показане, як виглядає проект веб-розробки MEAN, що виходить за рамки простого синтаксису, від початку й до кінця. Ця перша частина присвячена практичному введенню в технології компонентів стека, включаючи процеси їх установки й настроювання.

MEAN знаменує собою важливе зрушення в області архітектури й способу мислення в сфері програмування — від реляційних баз даних до Nosql і від схеми « модель-вистава-контролер» на стороні сервера до клієнтських односторінкових додатків. MEAN — це більше, ніж проста перестановка початкових букв і нових технологій. Зсув базової платформи з ОС (Linux) до середовища виконання Javascript (Node.js) несе із собою незалежність від ОС: Node.js працює на Windows® і OS X так само, як і на Linux.

Node.js заміняє Apache зі стека LAMP. Але Node.js — це набагато більше, чим просто веб-сервер. Насправді готовий додаток не розгортається на окремому веб-сервері; замість цього сам веб-сервер включається в додаток і автоматично встановлюється в складі стека MEAN. У результаті процес розгортання значно спрощується, тому що необхідна версія веб-сервера явно визначена разом з іншими залежностями часу виконання.

Yeoman: комбінація із трьох інструментів розробки на основі командного рядка для побудови каркаса (Yo),створення сценаріїв (Grunt)і керування залежностями на стороні клієнта (Bower).

Bootstrap: бібліотека CSS, яка забезпечує адаптивний веб-дизайн для

поддержки мобільних обладнань.

Бібліотеки тестування: крім Mocha, Jasmine і Karma, ціла плеяда бібліотек тестування, призначених для імітації викликів Ajax (Chai),демонстрації тестового покриття (Istanbul)і автоматизації функціональних тестів для роботи в реальних браузерах (Protractor).

Перехід від традиційної бази даних, такий як Mysql, до бессхемному, документо-орієнтованому Nosql-сховищу, такому як Mongodb, являє собою фундаментальне зрушення в стратегії персистенции. Програміст витрачає менше часу на написання операторів SQL і більше – на написання функцій map/reduce на Javascript. При цьому виключаються величезні шари логіки перетворення, тому що Mongodb споконвічно видає формат JavaScript Object Notation (JSON). У результаті гранично спрощується написання веб-сервісів REST.

Але головне зрушення між LAMP і MEAN полягає в переході від традиційного генерування сторінок на стороні сервера до орієнтації на односторінкові додатки (SPA) на стороні клієнта. Express дозволяє управляти й маршрутизацією / генерацією сторінок на стороні сервера, але тепер — завдяки Angularjs — упор робиться на вистави на стороні клієнта. Ця зміна означає не просто перенос ваших артефактів модель-вистава-контролер (MVC) із сервера в клієнтське обладнання. Це також стрибок від менталітету синхронності до менталітету, що носить подійно-керований, принципово асинхронний характер. І, мабуть, найголовніше, – це рух від сторінко-орієнтованих додатків до компонентно-орієнтованих.

Стік MEAN не «заточений» на мобільні додатки — Angularjs однаково добре працює на настільних комп'ютерах і ноутбуках, смартфонах і планшетах і навіть на смарт-телевізорах, — але він і не ставиться до мобільних обладнань як до громадянам другого сорту. І тестування більше не відкладається на потім: за допомогою платформ тестування світового класу, таких як MochaJS,JasmineJS і KarmaJS, можна писати ретельні й всеосяжні набори тестів для своїх Mean-Додатків.

Команда nvm ls показує, які версії Node.js у вас уже встановлені і яка версія

використовується в цей час.

На момент написання цієї статті на веб-сайті Node у якості останньої стабільної версії була зазначена версія v0.10.28. Уведіть команду nvm install v0.10.28, щоб установити її локально.

Після установки Node.js (через NVM або установником для конкретної платформи) уведіть команду node--version, щоб переконатися, що ви використовуєте поточну версію:

1.3 Аналіз і застосування Node.js

Node.js — це автономне середовище виконання Javascript. Практично це той же механізм Javascript (називаний V8), який працює в Google Chrome, за винятком того, що Node.js дозволяє запускати Javascript з командного рядка, а не із браузера.

Для зручності можна використовувати інструменти розробки у своєму звичному браузері.

Javascript уперше з'явився у веб-браузері (Netscape Navigator 2.0).

У мови програмування Javascript немає вбудованих можливостей для маніпуляцій з Document Object Model (DOM) або для створення запитів Ajax. Браузери забезпечують Api-Інтерфейси DOM, що дозволяє Javascript робити такого роду речі, але поза браузером Javascript втрачає цю здатність.

Відповіддю на команду navigator.appname служить Netscape, а відповіддю на команду navigator.appversion — загадковий рядок агента користувача, знайома досвідченим веб-розроблювачам. На малюнку 1 (скриншот Chrome на OS X) цей рядок виглядає так: 5.0 (Macintosh; Intel Mac OS X 10_9_4) Applewebkit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36.

Рисунок 1.4 – Використання об'єкта navigator Javascript у веб-браузері

Тепер створіть файл із іменем test.js. Уведіть ті ж команди у файл, упакувавши кожну у виклик console.log():

– console.log(navigator.appname);

– console.log(navigator.appversion);

Збережете файл і введіть команду node test.js, щоб запустити його, як показано .

Повідомлення про помилку в Node.js:

```
 navigator is not defined
$ node test.js

/test.js:1
ion    (exports,    require,    module,    __filename,    __dirname)    {
console.log(navigator.
                                                                    ^

Referenceerror: navigator is not defined
    at Object.<anonymous> (/test.js:1:75)
    at Module._compile (module.js:456:26)
    at Object.Module._extensions..js (module.js:474:10)
    at Module.load (module.js:356:32)
    at Function.Module._load (module.js:312:12)
    at Function.Module.runmain (module.js:497:10)
    at startup (node.js:119:16)
    at node.js:902:3
```

Згідно із трасуванням стека, не завантажується потрібний модуль. (Модулі — це ще одна важлива відмінність між роботою Javascript у браузері й в Node.js).

В Javascript можна створювати спеціалізовані функції, але — на відміну від

Java, Ruby або Perl — немає можливості об'єднати кілька функцій у єдиний модуль або «пакет», який можна імпортувати й експортувати. Звичайно, будь-який вихідний файл Javascript можна включити за допомогою елемента <script>, але цей перевірений часом метод уступає нормальній декларації модуля по двом важливим статтям.

По-перше, будь-який Javascript, включений за допомогою елемента <script>, завантажується в глобальний простір імен. З модулями можна обмежити імпортовані функції змінної в локальному просторі імен. По-друге, і це головне, модулі дозволяють явно оголосити залежності, а елемент <script> — немає. У результаті при імпорті модуля A транзитивно імпортуються залежні модулі B і C. У міру ускладнення додатків можливість керування транзитивними залежностями швидко стає життєво важливою вимогою.

Проект Commonjs, як і припускає назва, визначає загальний формат модулів (у числі інших специфікацій Javascript поза браузером). Node.js являє собою одну з багатьох існуючих реалізацій Commonjs. Ringojs (сервер додатків, аналогічний Node.js, який працює в середовищі виконання Javascript JDK Rhino/Nashorn) теж заснований на Commonjs, як і популярні сховища персистенции Nosql Couchdb і Mongodb.

Модулі – особливість наступної основної версії Javascript, але до широкого впровадження цієї версії Node.js використовує свою власну версію модулів на основі специфікації `CommonJS`.

Включите модуль Commonjs у свій сценарій за допомогою ключового слова require. Наприклад, Hello World в Node.js

```
var http = require('http');
var port = 9090;
http.createserver(responsehandler).listen(port);
console.log('Server running at http://127.0.0.1:' + port + '/');

function responsehandler(req, res){
  res.writehead(200, {'Content-Type': 'text/html'});
  res.end('<html><body><h1>Hello World</h1></body></html>');
}
```

Усі сотні (або тисячі) раз писали інструкції типу var port = 9090;. Ця інструкція визначає змінну з іменем port і привласнює їй значення 9090. Імпорт модуля Commonjs, як це зроблене в першому рядку (var http = require('http');), нічим не відрізняється. Він уводить модуль http і призначає його локальної змінної. Усі супутні модулі, на які опирається http, також викликаються оператором require.

Наступні рядки example.js виконують наступні дії.

– створення нового HTTP-сервера;
– призначення функції для обробки відповідей;
– запуск перехоплювача вхідних запитів HTTP для зазначеного порту.

Так – усього в декількох рядках Javascript — ми створили в Node.js простий веб-сервер. Як ви довідаєтеся з наступних руководств цієї серії, Express доповнює цей простий приклад обробкою більш складних маршрутів і обслуговуванням як статично, так і ресурсів, що динамічно генерируються.

Модуль http – це стандартна частина будь-якої установки Node.js. У число інших стандартних модулів Node.js входять модулі введення-висновку файлів, читання даних, що вводяться користувачем у командному рядку, низькорівневих запитів TCP і UDP і багато інші. У розділі «Модулі» документації Node.js утримується повний список стандартних модулів і опис їх можливостей.

Хоча цей список вбудованих модулів вражає, він блідне в порівнянні зі списком сторонніх модулів. Щоб одержати до них доступ, потрібно познайомитися з іншою утилітою командного рядка: NPM.

NPM — розшифровується як Node Packaged Modules. На веб-сайте NPM є список більш ніж з 75 000 загальнодоступних сторонніх модулів Node. Знайдіть на цьому сайті модуль yo. Результати показано на рисунку 1.5.

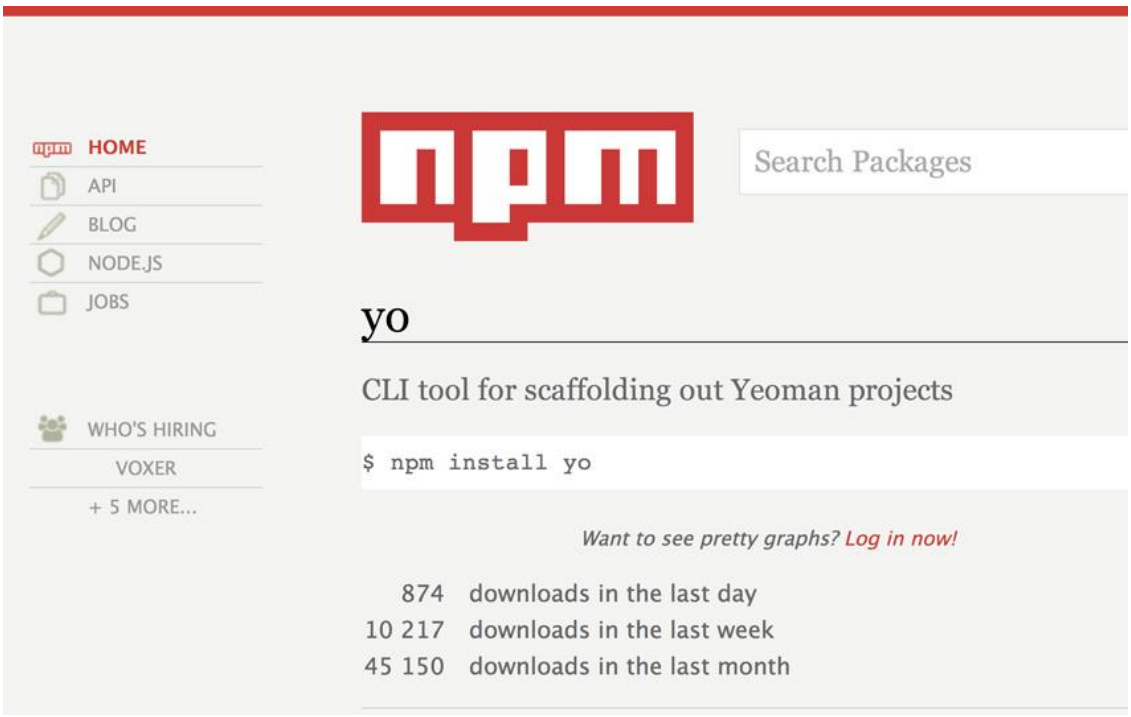

Рисунок 1.5 – Докладна інформація про модуль yo

Сторінка результатів містить короткий опис модуля («Інструмент CLI для підготовки структури проектів Yeoman»); кількість завантажень за минулий день, тиждень і місяць; ім'я автора; залежні модулі (якщо є) і багато чого іншого. Найголовніше, що сторінка результатів дає синтаксис командного рядка для установки модуля.

Щоб одержати аналогічну інформацію про модуль yo з командного рядка, уведіть npm info yo. (Якщо ви ще не знаєте офіційної назви модуля, можна ввести npm search yo для пошуку будь-якого модуля, ім'я якого містить рядок yo.) Командаnpm info виводить уміст файлу package.json модуля.

З кожним модулем Node.js повинен бути зв'язаний коректний файл package.json, тому варто ближче познайомитися із умістом цього файлу. У лістингу показаний вміст файлу package.json для модуля yo, розділене на три частини.

Перші елементи, – це звичайно name, description і масив доступних версій versions у форматі JSON.

```
$ npm info yo
```

```
{ name: 'yo',
  description: 'CLI tool for scaffolding out Yeoman projects',
  'dist-tags': { latest: '1.1.2' },
  versions:
   [
     '1.0.0',
     '1.1.0',
     '1.1.1',
     '1.1.2' ],
```

Щоб установити останню версію модуля, необхідно ввести npm install package. Команда npm installpackage@version установлює певну версію.

Далі, як показано в лістингу, випливають імена авторів, що супроводжують і адреса сховища Github, де перебуває вихідний код.

```
Package.JSON.
author: 'Chrome Developer Relations',
repository:
 { type: 'git',
   url: 'git://github.com/yeoman/yo' },
homepage: 'http://yeoman.io',
keywords:
 [ 'front-end',
   'development',
   'dev',
   'build',
   'web',
   'tool',
   'cli',
   'scaffold',
   'stack' ],
```

У цьому випадку також є посилання на головну сторінку проекту й Json-Масив ключових слів. Не в кожному файлі package.json присутні всі ці поля, але користувачі рідко скаржаться на занадто велику кількість метаданих, пов'язаних із проектом.

Номера версій відповідають загальному шаблону основна версія.проміжна версія.версія виправлення, який називається SemVer (Semantic Versioning).

```
engines: { node: '>=0.8.0', npm: '>=1.2.10' },
dependencies:
 { 'yeoman-generator': '~0.16.0',
   nopt: '~2.1.1',
   lodash: '~2.4.1',
   'update-notifier': '~0.1.3',
   insight: '~0.3.0',
   'sudo-block': '~0.3.0',
   async: '~0.2.9',
```

```
    open: '0.0.4',
    chalk: '~0.4.0',
    findup: '~0.1.3',
    shelljs: '~0.2.6' },
peerdependencies:
 { 'grunt-cli': '~0.1.7',
   bower: '>=0.9.0' },
devdependencies:
 { grunt: '~0.4.2',
   mockery: '~1.4.0',
   'grunt-contrib-jshint': '~0.8.0',
   'grunt-contrib-watch': '~0.5.3',
   'grunt-mocha-test': '~0.8.1' },
```

Цей файл package.json указує на те, що його потрібно встановлювати на екземпляр Node.js версії 0.8.0 або вище. А якщо ні, то команда `npm install` не виконається.

Скорочений синтаксис Semver – знак тильда (~) у багатьох версіях залежностей. Це еквівалентно 1.0.x (також припустимий синтаксис), що означає «основна версія повинна бути 1, проміжна версія повинна бути 0, але можна встановити останню версію виправлення, яку ви знайдете». В Semver мається на увазі, що версії виправлень ніколи не вносять критичних змін в API (звичайно це виправлення для існуючих функцій), а проміжні версії вносять додаткову функціональність (наприклад, нові виклики функцій), не порушуючи існуючої.

Крім необхідної платформи, цей файл package.json також містить кілька списків залежностей:

– блок `dependencies` містить список залежностей часу виконання;

– блок `devdependencies` містить список модулів, необхідних у процесі розробки;

– блок peerDependencies дозволяє авторові визначити «колегіальні» зв'язки між проектами. Ця можливість часто використовується для вказівки зв'язку між базовим проектом і його плагинами, але в цьому випадку він указує на два інших проекту (Grunt і Bower), які становлять проект Yeoman поряд з Yo.

Якщо ввести команду npm install без імені модуля, то npm шукає в поточному каталозі файл package.json і встановлює всі його залежності, перераховані в три описані вище блоках.

Наступний крок до одержання діючого стека MEAN — установка Yeoman і відповідного генератора Yeoman-mean.

1.4 Види хмарних сервісів

Серед найбільш повних визначень хмарних систем на сьогоднішній день можна виділити наступні два:

– хмарні сервіси – це технологія обробки даних, у якій програмне забезпечення надається користувачеві як інтернет-сервіс, при якім від користувача схована інфраструктура «хмари» (хмарної системи) і, тому, йому не потрібні спеціальні знання й навички для керування й використання даної «хмарної» технології;

– хмарні обчислення – це обчислення, які являють собою динамічно масштабований спосіб доступу до зовнішніх обчислювальних ресурсів у вигляді сервісу, надаваного за допомогою Інтернету.

Дані визначення тісно зв'язані між собою: для реалізації всіх хмарних сервісів необхідні обчислення, а хмарні обчислення по суті самі є хмарним сервісом. Розглянемо класифікацію хмарних сервісів і їх реалізацій із світу вільного ПЗ.

У цей час усі хмарні сервіси підрозділяють на кілька категорій.

Програмне забезпечення як послуга (Softwareas a Service, скорочено Saas) – бізнес-модель продажу програмного забезпечення, при якій власник (постачальник) ПО надає доступ до нього користувачам (замовникам) через Інтернет. Прикладами такого ПО є Feng Office Community Edition, Simple Groupware, Zarafa і т.ін. [4].

Устаткування (обчислювальні потужності) як послуга (Hardware as a Service, скорочено Haas) – надання обчислювальних ресурсів устаткування (його процесорного часу, місця для зберігання даних і т.д.) у вигляді сервісів з

використанням технологій віртуалізації. Сервіси, звичайно пропонуються як еквівалент реальним обчислювальним системам, таким як сервери, суперкомп'ютери. Над програмною реалізацією цієї ідеї повністю або частково працюють проекти Open VZ, Free VPS, Linux-vserver, Apache Hama, Glusterfs Open Source Project, а також Moose File System (Moosefs) і ін., а надає такий сервіс на базі Open Source рішень компанія Linode і деякі інші.

Комунікація як Сервіс (Communications as a Service – Caas) – побудоване в хмарі комунікаційне рішення для підприємства, яке забезпечує передачу мовного сигналу по мережі Інтернет або по будь-яких інших Ip-Мережам (Voip), обмін миттєвими повідомленнями (IM), відеоконференції. Модель Caas дозволяє діловим клієнтам вибірково розвертати засоби комунікацій і послуг на підставі оплати послуг у строк для використовуваних сервісів. Із цим напрямком тісно зв'язані такі Foss-Проекти як Ekiga, ilbc, Speex.

Моніторинг як Сервіс (Monitoring as a Service, скорочено Maas) є, що обслуговується в хмарі програмним забезпеченням для моніторингу й забезпечення безпеки. Такими Open Source-Рішеннями на сьогоднішній день є Ganglia, Zabbix, Hyperic HQ. Сюди ж з деякими застереженнями модно віднести й Nagios.

Інфраструктура як послуга (Infrastructure as a Service, скорочено Iaas) – це надання комп'ютерної інфраструктури (як правило, у формі віртуалізації) як послуги на основі концепції хмарних обчислень. По суті Iaas є комбінацією Saas, Haas, тому що вона містить у собі й те й інше, причому, звичайно в множині, а також Caas і, іноді, Maas з метою об'їдання й моніторингу всієї системи, і, тому, використовується в основному підприємствами. Вільними реалізаціями даної концепції є Eucalyptus, Opennebula, Openstack, Nimbus і ін.

Платформа як послуга (Platform as a Service, скорочено Paas) – надання програмної платформи й інструментів з певними характеристиками, необхідних для розробки, тестування, розгортання, підтримки різних додатків. Сюди ж входять і готові до використання хмарні сервіси, які разом утворюють програмну платформу. Яскравими прикладами з миру Open Source у цей час є Xen Cloud

Platform, Cloud Foundry, Apache Hadoop, Apache Hive і ін.

Комп'ютер (віртуальний робітник стіл) як послуга (Desktopas a Service, скорочено Daas) – надання віртуального комп'ютера, який каждий користувач може індивідуально набудовувати під свої завдання. Таким чином, користувач, приходячи на роботу, просто вводить свої дані (звичайно логін і пароль) і може працювати, використовуючи при цьому завдяки технологіям віртуалізації обчислювальні потужності стороннього сервера, а не свого ПК.

Робоче оточення як послуга (Workspaceas a Service, скорочено Waas) – надання комплекту Saas, призначеного для створення робочого оточення. На відміну від Daas у цьому випадку користувач одержує доступ тільки до ПО, у той час як усі обчислення відбуваються безпосередньо на його машині. По суті, дана категорія є гібридом Saas і Paas. На відміну від останньої Iaas є платформою, спрямованої не на розробку й тестування ПО, а на офісну роботу. Однак при цьому також як і Saas у реалізації не використовує технологій віртуалізації. На даний момент реалізації даної технології надаються в основному різними великими компаніями, наприклад Google і Microsoft, і представляють в основному рішення із закритим вихідним кодом, іноді з використанням вільних і відкритих компонентів або їх вихидних кодів.

Усі як послуга (Everything as a service, скорочено Eaas) – концептуальна модель, що включає в себе елементи всіх перерахованих рішень. На даний момент повної її реалізації не існує – вона, по суті, є ідеалом для великих хмарних компаній, таких як Google і Microsoft [5].

Інфраструктура як Сервіс – Iaas у значній мірі підсилює технологію, послуги й вкладення в Цоди, щоб надати це як послугу клієнтам. На відміну від традиційного аутсорсінгу, який вимагає певного ретельності, нескінченних переговорів і складних контрактів, Iaas зосереджена навколо моделі надання послуг, яка забезпечує визначену, стандартизовану інфраструктуру, що враховує потреби клієнта. Спрощені пропозиції роботи й вибір рівня обслуговування спрощують клієнтові вибір рішення з регламентованим набором стандартних і необхідних експлуатаційних характеристик. Як правило, постачальники надають

компоненти наступних рівнів:

– комп'ютерна мережа (включаючи маршрутизатори, брандмауерів, балансування навантаження і т.ін.);

– платформа віртуалізації для того, щоб запускати віртуальні машини;

– апаратне забезпечення (зазвичай це Грід з масивною горизонтальною масштабованістю);

– угоди сервісного обслуговування;

– підключення Інтернет;

– платформа віртуалізації для того, щоб запускати віртуальні машини;

– інструменти обліку обчислень.

Клієнти, зацікавлені даним хмарним рішенням не здобувають сервера, мережне встаткування, спеціалізоване ПО, а тільки орендують ресурси, які належать обслуговуючим постачальникам послуг Iaas. Клієнт оплачує тільки споживані їм ресурси, що дуже вигідно.

Основні переваги цього хмарного сегмента включають:

– мінімізація ризиків, за рахунок використання вилучених ресурсів, підтримуваних третіми особами;

– використання інфраструктури останнього покоління;

– здатність управляти піковими навантаженнями;

– вільний доступ до попередньо сконфигурованого навколишнього середовища;

– більш низькі витрати;

– захищені й ізольовані обчислювальні платформи;

– легка розширюваність платформи.

– менші витрати часу й засобів, для додавання або розширення функціональності.

Платформа як сервіс (Paas) . Для розгортання веб-додатків розроблювачеві немає необхідності купувати встаткування, програмне забезпечення, пет необхідності організовувати їхню підтримку. Клієнт може одержати доступ до

додатка на права оренди [6].

Такий підхід має наступні переваги:

– масштабованість;

– віртуалізація;

– стійкість до відмов;

– безпека.

Масштабованість Paas припускає автоматичне звільнення й виділення запитуваних ресурсів залежно від кількості користувачів, що обслуговуються додатком. Paas – як інтегрована платформа для тестування, створення, розгортання й підтримки веб-додатків, здатна забезпечити виконання всього переліку операцій для веб-додатків, в одному інтегрованім середовищі. Виключаються витрати на підтримку окремих середовищ, для окремих етапів.

Можливість створення вихідного коду й передача його іншим розроблювачам даного додатка значно підвищує продуктивність діяльності по створенню додатків на основі Paas.

Прикладом даного виду хмарних сервісів є Appengine від Google, що пропонує хостинг для веб-додатків, а також з можливим збільшенням обчислювальні ресурси (ідеально підходить, наприклад, для тестування високих навантажень). Для виконання додатків Google Appengine була розроблена платформа Appscale, яка, однак, належить не Google.

Центральне місце в Paas займає операційна система Microsoft Windows Azure. Windows Azure створює єдину платформу, що включає хмарні аналоги продуктів Microsoft (реляційна база даних SQL Azure – аналог SQL Server, а крім цього Exchange Online, Share Point Online і Microsoft Dynamics CRM Online) і інструменти розробки (.NET Framework і Visual Studio, оснащена з версії 2010 року набором Windows Azure Tools). Так, користувач, що розробив сайт в Visual Studio, може, не залишаючи додатка помістити створений сайт в Windows Azure.

1.5 Обґрунтування цілей досліджень

Інтеграція корпоративних додатків і даних завжди займала перші позиції в списку пріоритетних і одночасно найбільш складних завдань підрозділів IT, що займаються супроводом безлічі прикладних систем, що підтримують різні бізнес-процеси. Штатне функціонування всіх таких процесів вимагає злагодженої спільної роботи часом відразу декількох додатків, і не випадково в побут IT-фахівців міцно ввійшло поняття інтеграційного «спагетті», що характеризує складність логічної картини, яка, як правило, складається в спробі розв'язати всі варті перед компанією завдання інтеграції.

Основна ідея обчислень у хмарах полягає в тому, щоб дозволити компаніям одержувати послуги провайдерів на умовах аутсорсингу. Більшість компаній віддадуть перевагу не впровадженню власної хмари, а звертання до провайдера. У міру росту попиту на такі послуги, будуть збільшуватися витрати компаній і обсяг супровідних робіт. Зрештою, це може привести до того, що компанія виявиться «прив'язаною» до певного провайдера й набору сервісів, тому що, вклавши засобу в розширення сервісів або їх інтеграцію, ви чи навряд захочете втратити ці вкладення, помінявши сервіси або провайдера.

Доступ до Saas здійснюється з використанням мережі через браузер. Перевага Saas – це можливість заощадити на установці, конфігурації, адмініструванні встаткування й установленого на ньому ПЗ.

Цільова аудиторія – кінцеві споживачі.

У моделі Saas:

– додаток пристосований для вилученого використання;

– можливість використання одного додатка декількома користувачами;

– послуга оплачується щомісячно, або на основі отриманих транзакцій;

– підтримка додатка включена до складу оплати;

– удосконалювання додатка може проводитися плавно й прозоро для клієнтів.

Модель Saas дозволить більш ефективно боротися із програмним піратством, оскільки неможливе копіювання, поширення й установка клієнтом даного програмного забезпечення. По суті, модель Saas є якісною альтернативою внутрішнім інформаційним системам.

Рішенням цієї проблеми стане звертання до хмарних брокерів, які нададуть послуги посередництва або агрегування хмарних сервісів, що дозволяють компанії одержати те, що їй потрібно. Володіючи відповідною кваліфікацією, вони відіграють роль, аналогічну системним інтеграторам, тому що забезпечують багатьом замовникам керування хмарними сервісами. У такому випадку компанія не буде прив'язана до одного хмарного провайдера, тому що буде платити не провайдерам, а брокерам.

2 ОПИС ПРОВЕДЕНИХ ТЕОРЕТИЧНИХ ДОСЛІДЖЕНЬ

2.1 Поняття інтеграційної платформи

Найбільш яскравою тенденцією останнього часу з погляду використання й інтеграції корпоративних додатків стала поява так званих рішень типу системи залучення (systems of engagement), які протиставляються традиційним системам обліку (systems of record). Останні – це класичні транзакційні системи корпоративного бек-енду, такі як ERP, системи фінансового обліку, традиційні рішення по керуванню персоналом і т.д. Системи залучення – це рішення нового типу для керування взаєминами із клієнтами, організації спільної роботи співробітників, керування талантами й ін. На відміну від транзакційних систем, що автоматизують операції певних департаментів компанії, системи залучення підтримують взаємозв'язки між співробітниками, клієнтами й партнерами й можуть використовувати для цього мобільні й соціальні механізми. Обоє типу корпоративних додатків сильно відрізняються по реалізації, але сьогодні їх необхідно інтегрувати, оскільки все частіше в бізнес-процесах компаній задіються й ті, і інші системи. Наприклад, рішення про надання знижки клієнтові може бути прийняте на основі даних хмарної системи клієнтської аналітики, які фахівець у компанії одержить, працюючи в корпоративній Erp-Системі.

Пошук ефективних способів для зв'язку систем залучення із традиційними бек-офісними рішеннями становить сьогодні ключову проблему інтеграції.

Справа в тому, що для того щоб одержати нову функціональність «систем залучення», компанії все частіше прибігають до готових Saas-Рішенням або використовують хмарні платформи для реалізації власних додатків такого типу, що вимагає механізмів інтеграції локальних і хмарних систем. Крім нетривіальності самого цього завдання, ситуацію ускладнює проблема швидкості відновлень хмарних рішень. Якщо установка нових версій, наприклад, корпоративної ERP-платформи завжди була під контролем керівника IT-служби, який знав періодичність таких відновлень і управляв процесом, то нова

функціональність Saas-рішень або хмарних платформ, як правило, з'являється кожні кілька тижнів і автоматично впроваджується в продуктивну експлуатацію. Збільшує ситуацію той факт, що до Saas-додаткам бізнес-підрозділи часом прибігають по власному розсуду, не погоджуючи це рішення з корпоративними IT-політиками.

Таким чином, інтеграційне середовище сьогодні повинна мати можливість роботи з гібридними рішеннями, що поєднують локальні й хмарні компоненти. За даними дослідження Forrester, в компаніях у середньому використовується близько десяти Saas-систем, близько чверті компаній застосовували для різних етапів одного бізнес-процесу локальні й хмарні системи, причому використання Saas росло не тільки для нових типів додатків, але й для традиційних систем бек-офісу. І ключова тенденція тут полягає в тому, що хмарні додатки, стаючи для бізнесу усе більш звичними й часто кращими, не витісняють, а доповнюють локальні рішення.

Поточний стан прикладного IT-ландшафту компаній аналітики характеризують як гібридне — бізнес-процеси підтримуються конгломератом локальних, мобільних і хмарних додатків. Відповідно, підходи до їхньої інтеграції повинні враховувати цю гібридну природу. Ключовими можливостями платформи такої інтеграції є керування життєвим циклом метаданих і інтероперабельність у реальному часі, що дозволить IT-менеджерам сформувати з безлічі інтеграційних продуктів керовану платформу інтеграції, але при цьому гнучку й здатну швидко мінятися.

По даним Forrester, сьогодні в компаніях у середньому використовуються інтеграційні рішення не менш чому трьох різні постачальників, плюс з'являються нові хмарні системи, призначені для рішення завдань інтеграції. Недолік сумісності між існуючими інструментами інтеграції й необхідність в узгодженні інтеграції локальних і хмарних додатків, а також в ув'язуванні між собою використання локальних і хмарних засобів інтеграції є, на думку аналітиків, основними перешкодами на шляху ефективної інтеграції в сучасному корпоративному IT-ландшафті, і саме на їхнє усунення спрямована концепція

гібридної інтеграції. [16]

Платформа гібридної інтеграції повинна підтримувати різноманітні інтеграційні інструменти, використовувані сьогодні в компаніях: системи ESB, шлюзи B2B для комунікацій із зовнішніми партнерами, класичні засоби інтеграції даних категорії ETL (extract, transform, load), інструментарій BPM, а також комплексні рішення (comprehensive integration solutions, CIS), що поєднують кілька класів інтеграційних засобів. Сьогодні інтеграційне середовище поповнюється системами нового класу — інструментами хмарної інтеграції, у яких аналітики бачать найбільший потенціал для інновацій, розділяючи інструменти інтеграції «із хмарою» і «у хмарі». Перші пропонують сконфигуровані пакети для взаємодії локальних систем інтеграції с Saas-Додатками або рішеннями, самостійно розробленими в хмарній інфраструктурі. Наприклад, платформа webmethods Cloudstreams компанії Software AG забезпечує таку інтеграцію для локально розміщених шин ESB.

Серед рішень для інтеграції «у хмарі» аналітики виділяють прості сервіси інтеграції на базі хмари (Cloud-Based Integration, CBI) і більш складні рішення типу integration-centric Platform-as-a-service (ipaas). Сервіси CBI дозволяють без зусиль підключати локальні системи до Saas-Додаткам, забезпечуючи, наприклад, необхідне первинне завантаження даних і їх наступну синхронізацію. Рішення такого типу реалізовані, наприклад, у продуктах Dell Boomi Atomsphere, IBM Cast Iron і Informatica C loud.

Хмарні рішення типу ipaas — це не просто перенос у хмарну інфраструктуру локальної системи інтеграції, наприклад ESB, а створення комплексної мультіарендної хмарного середовища типу Paas, що надає можливість створити, протестувати й увести в експлуатацію різні сценарії інтеграції бізнес-додатків, у тому числі локальних додатків з Saas. У якості прикладів ipaas можна назвати інтеграційний хаб Cloudhub компанії Mulesoft, систему Tibco Cloud Bus, сервіси інтеграції хмарних платформ Microsoft Azure і Red Hat Openshift.

Системи типу CBI і ipaas аналітики рекомендують зробити центральними в

інтеграційній стратегії компаній, де Saas і інші хмарні рішення витісняють локальні додатки для різних напрямків бізнесу, включаючи не тільки нові «системи залучення», але й традиційні системи корпоративного бекенду.

Крім хмарної інтеграції, зовсім нові завдання перед компаніями ставить парадигма Інтернету речей – системи ERP, клієнтська аналітика, CRM нового типу й інші корпоративні додатки можуть зажадати ефективному зв'язку з різними датчиками й обладнаннями. Засобу інтеграції для Інтернету речей починають з'являтися як незалежні нішеві рішення (наприклад, продукти німецької компанії Bosch Software Innovations для автомобільної індустрії) або нові інструменти в рамках існуючих інтеграційних пакетів (такі можливості з'являються, наприклад, у рішеннях Microsoft і Tibco Software).

Слід зазначити також важливість підтримки відкритих стандартів для забезпеченні консолідації різних інтеграційних інструментів. Звичайно, платформа інтеграції, побудована на базі декількох систем одного виробника, може виявитися ефективної завдяки використанню спеціалізованих пропрієтарних механізмів для їхнього зв'язку між собою, однак, з урахуванням різнорідного набору інструментів інтеграції в багатьох компаніях, можливість з'єднати в одне ціле рішення різних постачальників здобуває для корпоративних IT усе більше значення. Таку можливість простіше всього буде реалізувати, опираючись на індустріальні стандарти.

Аналітики підкреслюють, що ключову роль у забезпеченні інтероперабельності різнорідних систем інтеграції відіграє погоджене керування життєвим циклом метаданих. Платформа гібридної інтеграції повинна надавати можливість погоджувати між собою розробки й модифікації метаданих різних систем інтеграції додатків і даних, включаючи моделі процесів, моделі інфраструктури, таблиці відображення даних, конфігурації адаптерів, визначення сервісів, B2 B-Контракти, конфігурації EDI, визначення сервісів і т.д. Це дозволить знизити витрати на інтеграцію й зробить середовище інтеграції більш гнучкої, здатної швидко реагувати на зміни в додатках і поєднувати інструменти інтеграції додатків і даних. [17]

2.2 Аналіз хмарних сховищ за допомогою технологічних стеків

Зберігаючи дані в інтернеті головними проблемами, пов'язаними із захистом інформації й обмеженням доступу до хмарних ресурсів, є:

Конфіденційність – один з найважливіших аспектів ризиків безпеки при зберіганні й пересиланню даних у мережі Інтернет, тобто це гарантія того, що дані не стали доступними для сторонніх осіб.

Доступність – це можливість за прийнятний час одержати необхідну інформаційну послугу, тобто одержати доступ до даних на хмарних сховищах у будь-який час без затримок з будь-якого обладнання.

Цілісність – гарантія того, що дані будуть зберігатися правильно, що має на увазі актуальність і несуперечність інформації, її захищеність від руйнування й несанкціонованої зміни

Розглянемо основні переваги хмарних сервісів.

Можливість використання на малопотужних ПК. Для роботи з «хмарами» не потрібні дорогі комп'ютери з більшим обсягом пам'яті й дисководами, оскільки користувацька інформація зберігається на вилученому сервері. Споживачі можуть використовувати замість комп'ютерів і ноутбуків більш дешеві портабельні й зручні нетбуки. Єдиною обов'язковою умовою для раби є необхідність забезпечення безперебійного Інтернет- з'єднання [7].

Мінімізація витрат і збільшення продуктивності IT інфраструктури. Середнє значення завантаженості серверів компаній 10-15 **%.** Існують періоди, коли обчислювальна потужність необхідна, але іноді сервери простоюють. Якщо компанія використовує потрібну кількість ресурсів «хмарних сервісів», то її витрати на встаткування і його обслуговування знижуються на 50 %. Крім того в значній мірі збільшується гнучкість виробництва в динамічній економічній обстановці. Якщо якась велика корпорація побоюється за схоронність конфіденційних даних, які розташовані на серверах третіх осіб, то вона може дозволити собі власне «хмара» і насолоджуватися всіма перевагами від

віртуалізації інфраструктури.

Легкість і простота в обслуговуванні. При впровадженні Cloud Computing кількість фізичних серверів зменшується, що дозволяє швидше й легше їх набудовувати й обслуговувати. Крім того пропадає необхідність стежити за працездатністю ПЗ, оскільки актуальна його версія вже встановлена, настроєна й перебуває в «хмарі».

Регульована обчислювальна потужність. Її розмір обмежений лише можливостями хмари, тобто сукупністю великої кількості вилучених серверів, а не використовуваним комп'ютером. Це дає можливість рішення більш складних завдань, які вимагають надання більших обсягів пам'яті, для розташування даних у випадку такої необхідності. Можна сказати що, користувачі мають право використовувати супер-комп'ютер без яких-небудь додаткових вкладень.

Необмежене сховище даних. У порівнянні з комп'ютером фіксований обсяг, що має, доступного місця, «хмарне сховище» можливе підбудувати під різні потреби користувача. При зберіганні даних на вилучених серверах можна забути про кількість розташовуваної інформації, оскільки обсяг «хмарних» сховищ мільярдами гігабайт вільного місця.

Можливість роботи при наявності будь-якої операційної системи. В Cloud Computing операційна система не важлива. Це стирає всі обмеження між операційними системами, наприклад користувачі Unix можуть поділитися документами із власниками Microsoft Windows або навпаки, при цьому не випробовуючи яких-небудь незручностей.

Поліпшена сумісність форматів документів. Якщо користувачі працюють із тим самим «хмарним» програмним забезпеченням для створення й редагування документів, у них не може бути несумісних версій і форматів, на відміну від тих, хто одержить документ Word 2007 і зіштовхнеться із проблемою відкриття його за допомогою Word або Openoffice [8].

Зручність спільної роботи користувачів. При обробці документів немає необхідності їх пересилання між групою з метою перегляду змін, оскільки внесена зміна миттєва відбивається в інших користувачів. Доступ до документів у

будь-якому місці з будь-якого обладнання. Якщо документи перебувають в «хмарі» то доступ до них можна одержати з будь-якого обладнання при наявності Інтернет-З'єднання.

Безпека даних у випадку крадіжки або втрати встаткування. Поміщені в «хмару» дані, з метою забезпечення їх схоронності тиражуються й розподіляються по декільком серверам, які іноді перебувають друг від друга дуже далеко. При поломці комп'ютера інформація користувача не постраждає й буде доступна з будь-якого комп'ютера.

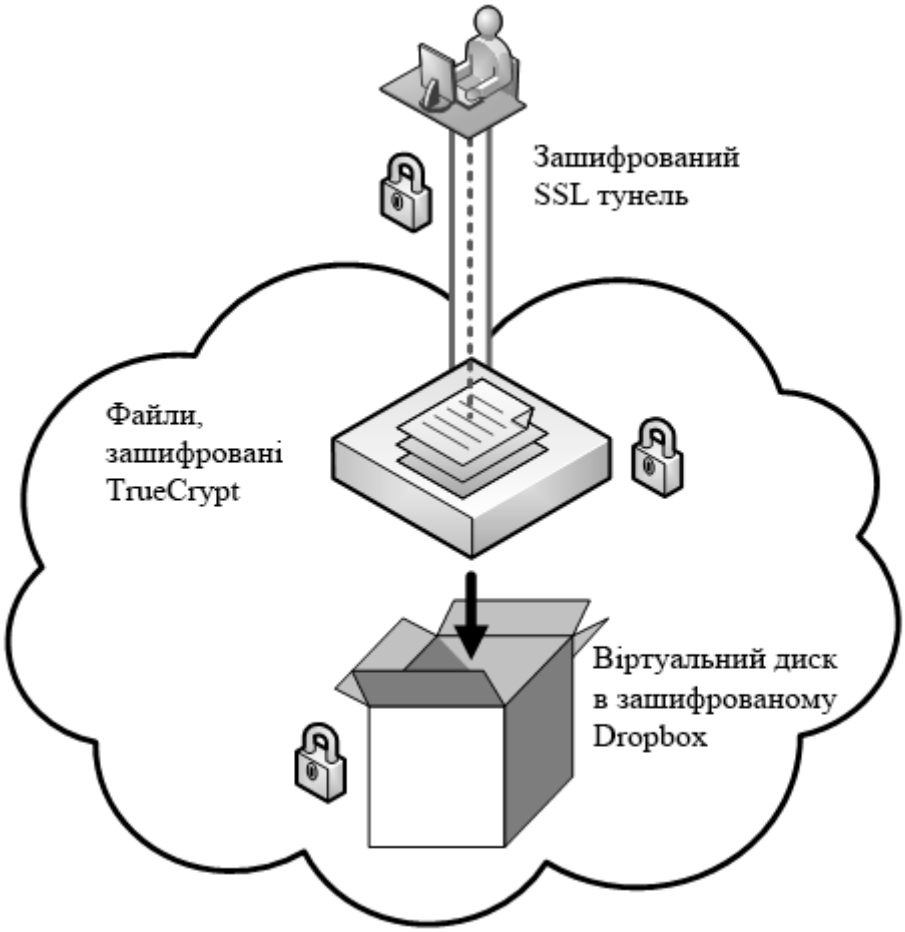

Рисунок 2.1 – Використання Truecrypt у сховище Dropbox

Зміни також можуть бути виявлені за допомогою цифр ASCII коду, які визначають букви, і в такий спосіб створюють секретне повідомлення (рисунок 2.2).

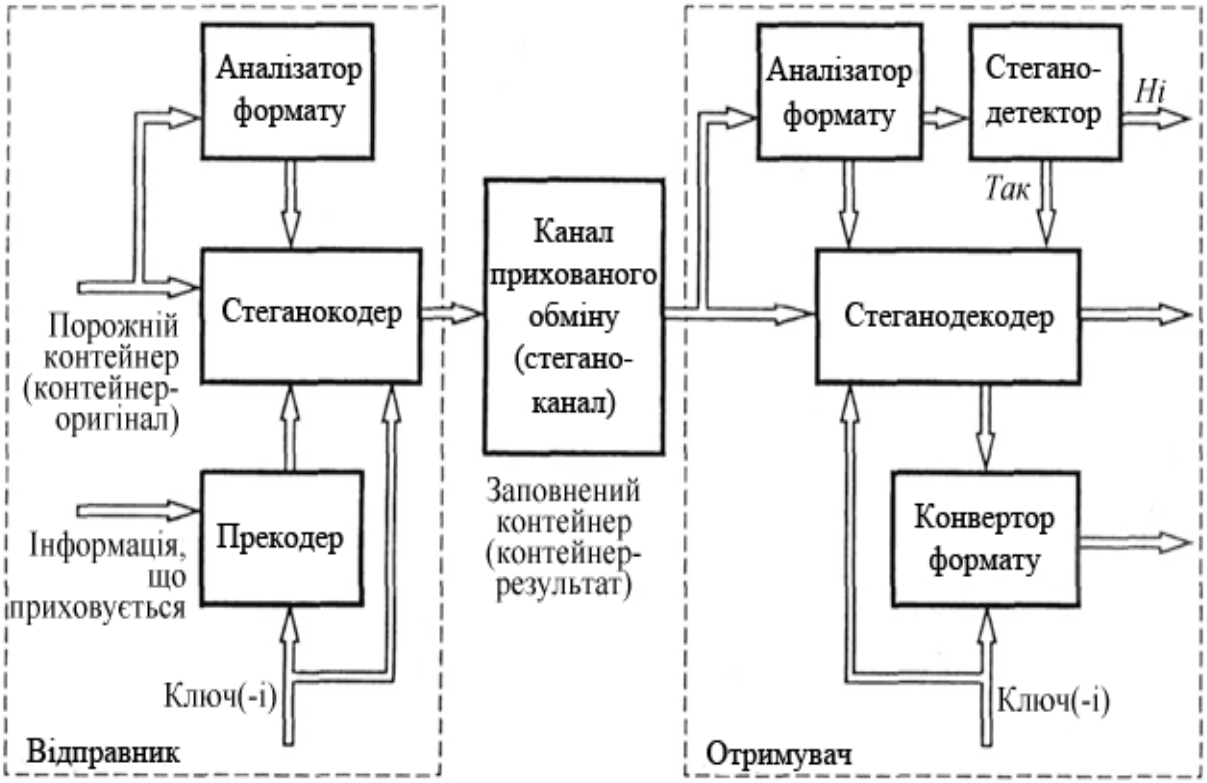

Рисунок 2.2 – Структурна схема стеганографії

Хмарна архітектура зберігання даних — це, насамперед, доставка ресурсів зберігання даних на вимогу у високомасштабуємому і мультитенантному середовищу. Узагальнено хмарна архітектура зберігання даних являє собою зовнішній інтерфейс, який надає API для доступу до накопичувачів. У традиційних системах зберігання даних це протокол SCSI, але в хмарі з'являються нові протоколи. Серед них можна знайти зовнішні протоколи Web-Сервісів, файлові протоколи й навіть більш традиційні зовнішні інтерфейси. За зовнішнім інтерфейсом розташовується рівень проміжного ПЗ. Цей рівень реалізує ряд функцій, таких як реплікація даних і скорочення обсягу даних, по традиційних алгоритмах розміщення даних (з урахуванням географічного розташування). Нарешті, внутрішній інтерфейс організує фізичне зберігання даних. Це може бути внутрішній протокол, який реалізує специфічні функції, або традиційний сервер з фізичними дисками.

Однією із ключових характеристик хмарної системи зберігання даних є її вартість. Якщо клієнт зможе купувати й управляти ресурсами зберігання

локально, а не орендувати їх у хмарі, ринок хмарних систем зберігання зникає. Його витрати можна розділити на дві загальні категорії: вартість самої фізичної екосістеми зберігання й витрати на керування нею. Вартість керування схована, але являє собою довгостроковий компонент загальної вартості. Із цієї причини хмарна система зберігання повинна бути в значній мірі самокерованої. Вирішальне значення має можливість додавати нові накопичувачі, коли система автоматично реконфігурується для їхнього розміщення, і здатність системи знаходити й автоматично виправляти помилки. У майбутньому такі концепції, як автономні обчислення, будуть відігравати ключову роль у хмарних архітектурах зберігання.

Одним із самих яскравих відмінностей між хмарної й традиційної системами зберігання є засоби доступу до них. Більшість постачальників пропонує різні методи доступу, однак загальноприйнятими є API Web-Сервісів. Багато хто з них реалізовані на принципах REST, що має на увазі об'єктно-орієнтовану схему, розроблену поверх HTTP (з використанням HTTP у якості транспорту). REST-API без запам'ятовування стану прості й ефективні. REST-API реалізують багато постачальників хмарних послуг зберігання, включаючи Amazon Simple Storage Service (Amazon S3), Windows Azure™ і Mezeo Cloud Storage Platform [18].

Одна проблема API Web-Сервісів полягає в тому, що для того щоб скористатися перевагами хмарної системи зберігання, вони вимагають інтеграції з додатком. Тому із хмарними системами зберігання для забезпечення безпосередньої інтеграції використовуються також загальні методи доступу. Наприклад, протоколи на основі файлів, такі як NFS/Common Internet File System(CIFS) або FTP, або протоколи на основі блоків, такі як iscsi. Такі методи доступу надають Nirvanix, Zetta, Cleversafe і інші постачальники послуг хмарного зберігання.

Вищезгадані протоколи найпоширеніші, але для хмарного зберігання підходять і інші. Один із самих цікавих — Web-based Distributed Authoring and Versioning (Webdav). Webdav також заснований на HTTP і дозволяє

використовувати Web у якості ресурсу для читання й запису. У число постачальників, що використовують Webdav, входять Zetta, Cleversafe і інші.

Можна знайти й такі рішення, які підтримують кілька протоколів доступу. Наприклад, IBM® Smart Business Storage Cloud дозволяє використовувати протоколи на основі файлів (NFS і CIFS) і протоколи на основі SAN в одній і тій же інфраструктурі віртуалізації систем зберігання даних.

Існує багато аспектів продуктивності, але головне завдання хмарної системи зберігання даних — це переміщення даних між користувачем і вилученим постачальником хмарних послуг. Проблема криється в TCP, головній робочій конячці Інтернету. TCP управляє потоком даних на основі підтвердження приймання пакетів з вилученого вузла. Втрата або затримка пакетів приводить до застосування заходів щодо обмеження скупчень пакетів з додатковим обмеженням продуктивності щоб уникнути глобальних мережних проблем. TCP ідеально підходить для переміщення невеликих обсягів даних через глобальну мережу Інтернет, але не для доставки більших обсягів даних – у цьому випадку час обміну даним (RTT) збільшується.

Працюючи зі стандартними мережними адаптерами (без прискорення), FASP ефективно використовує доступну додатку смугу пропущення й виключає головні вузькі місця традиційних схем масової передачі даних.

Одна із ключових особливостей хмарної архітектури зберігання називається мультітенантністю. Це означає, що сховище використовується багатьма користувачами або «орендарями». Мультітенантність торкається багато рівнів хмарної системи зберігання, від рівня додатка, де користувачам виділяються ізольовані простори імен, до рівня зберігання, де окремим користувачам або категоріям користувачів можуть виділятися окремі фізичні накопичувачі. Мультитенантность поширюється навіть на мережну інфраструктуру, яка з'єднує користувачів з накопичувачами, забезпечуючи гарантовану якість обслуговування й виділену смугу пропущення для конкретного користувача.

Рисунок 2.3 – Методи доступу до хмарних систем зберігання даних

Amazon за допомогою Aspera Software розв'язала цю проблему, виключивши з рівняння TCP. Для прискорення масового переміщення даних щоб уникнути більших RTT і великих втрат пакетів розроблений новий протокол Fast and Secure Protocol (FASP™). Ключем служить UDP, допоміжний транспортний протокол стосовно TCP. UDP дозволяє вузлу управляти заторами, передаючи цей аспект протоколу прикладного рівня FASP (рисунок 2.4).

Масштабованість можна розглядати з декількох точок зору, але нас в основному цікавить виділення хмарних ресурсів зберігання на вимогу. Можливість нарощувати ресурси зберігання (як нагору, так і вниз) означає поліпшену економічну ефективність для користувача й підвищену складність для постачальника хмарних послуг.

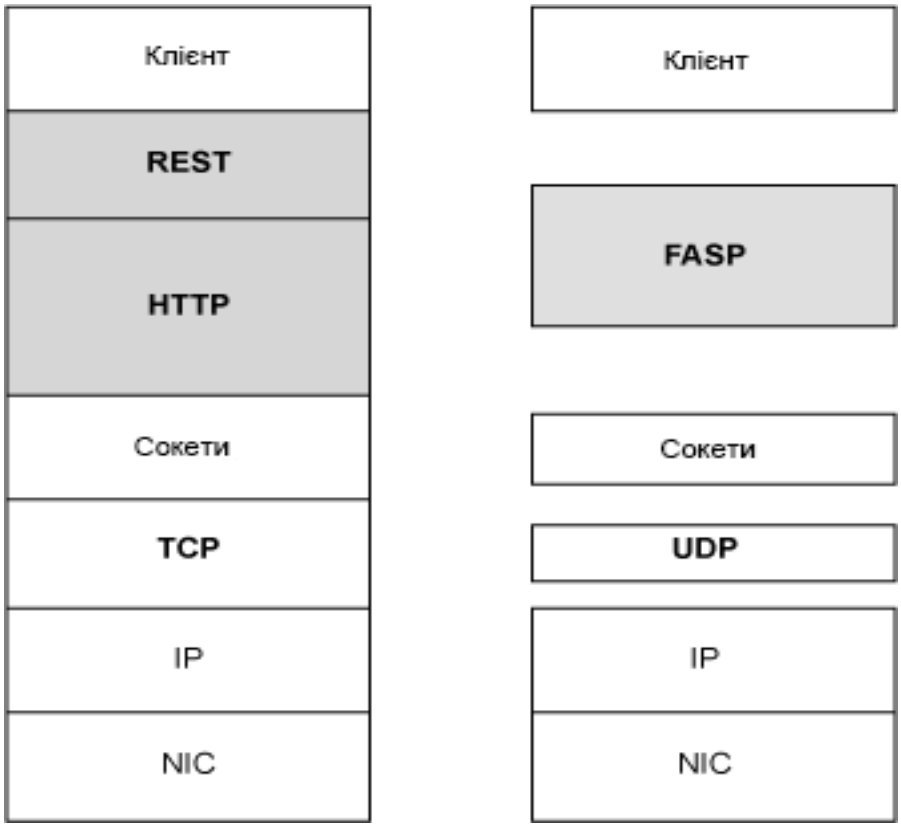

Рисунок 2.4 – Протокол Fast and Secure Protocol від Aspera Software

Масштабованість повинна забезпечуватися не тільки для самої системи зберігання (функціональне масштабування), але й для її пропускної здатності (масштабування навантаження). Ще одна ключова особливість хмарного зберігання — географічний розподіл даних (географічна масштабованість), яка дозволяє розташовувати дан у максимальній близькості до користувача завдяки групі центрів хмарного зберігання даних (шляхом міграції). У випадку даних "тільки для читання" можливі також реплікація й поширення (як у мережах поширення аудіо/відеоконтенту).

Хмарна інфраструктура зберігання повинна забезпечувати й внутрішнє масштабування. Сервери й система зберігання повинні допускати зміна розміру без усяких наслідків для користувачів.

Коли постачальник хмарних послуг зберігає дані користувача, він повинен мати можливість повернути ці дані користувачеві на вимогу. З урахуванням простоїв мережі, помилок користувачів і інших обставин виконання цієї умови надійним і детермінованим способом може виявитися скрутним [19].

Існують цікаві нові схеми забезпечення високої готовності, такі як розосередження інформації. Компанія Cleversafe, яка надає послуги зберігання даних у приватній хмарі, використовує алгоритм розосередження інформації (Information Dispersal Algorithm - IDA) для підвищення доступності даних перед особою фізичних відмов і простоїв мережі. Алгоритм IDA, спочатку розроблений для телекомунікаційних систем Майклом Рабином, дозволяє "нарізати" дані за допомогою кодів Рида-Соломона для їхнього відновлення у випадку втрати частини даних. Крім того, IDA дозволяє набудовувати кількість часток даних, так щоб заданий об'єкт даних можна було розрізати на чотири частки при одному припустимому збої або на 20 часток при восьми припустимих збоях. Як і RAID, IDA дозволяє відновлювати дані з підмножини вихідних даних при деяких накладних витратах на коди помилок (залежно від кількості припустимих збоїв) (рисунок 2.5).

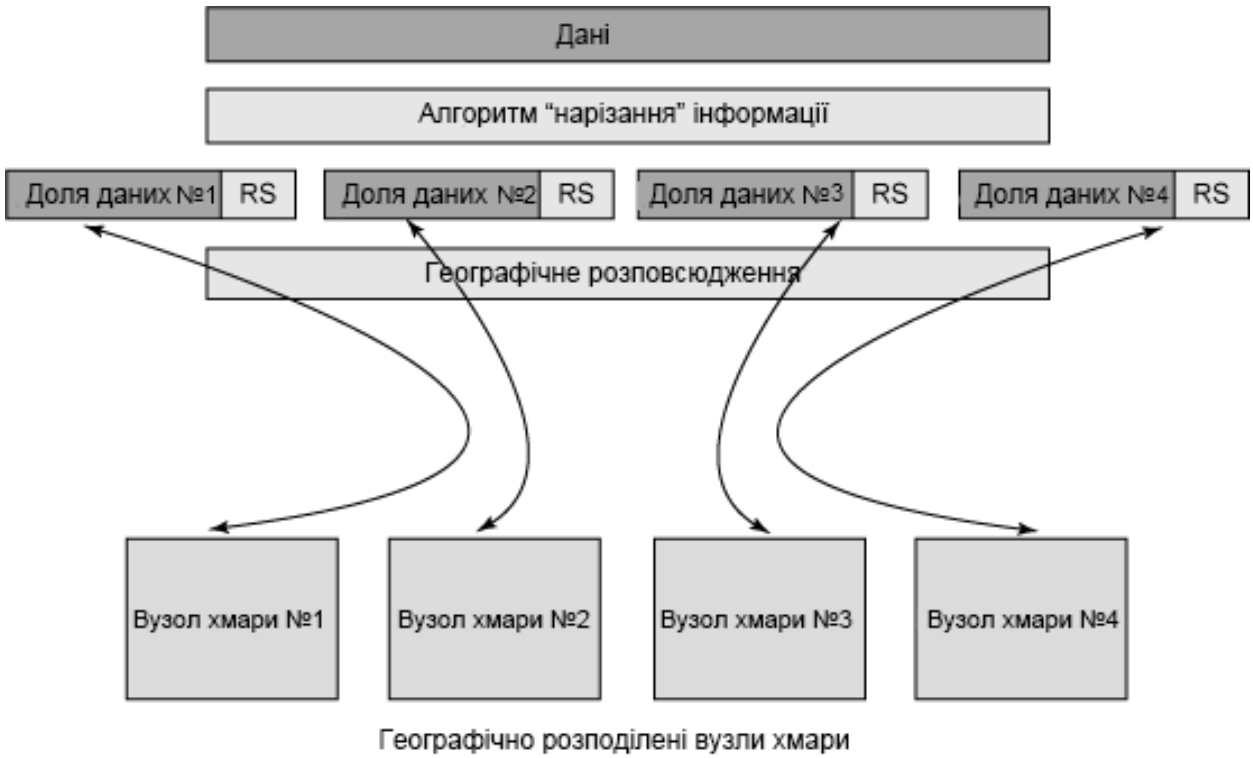

Рисунок 2.5 – Підхід Cleversafe до забезпечення високої готовності даних

Можливість нарізати дані із застосуванням кодів корекції Рида-Соломона дозволяє географічно розподіляти накопичувачі. При кількості часток p і припустимій кількості збоїв m результуючі накладні витрати становлять $p/(p-m)$.

Так, у випадку, показаному на рисунку 2.5, накладні витрати для системи зберігання при $p = 4$ і $m = 1$ становлять 33%.

Зворотний бік IDA – інтенсивна обробка без апаратного прискорення. Реплікація – ще один корисний метод, який використовують багато постачальників хмарних послуг. Він простий і ефективний, хоча й накладні витрати великі (100%).

Важливе значення має здатність клієнта контролювати й управляти тем, як зберігаються його дані, і пов'язаними із цим витратами. Численні постачальники хмарних послуг пропонують засобу керування, які забезпечують користувачам підвищений контроль над витратами.

Amazon, щоб надати користувачам засобу мінімізації загальних витрат на зберігання даних, застосовує Reduced Redundancy Storage (RRS). Дані репліціруются в інфраструктурі Amazon S3, але RRS дозволяє репліцирувати їх менша кількість раз із можливістю відновлення у випадку втрати даних. Це ідеально підходить для даних, які можна відтворювати, або коли копії даних розташовуються в різних місцях. Nirvanix також забезпечує реплікацію на основі правил, допускаючи більш детальний контроль над тем, де і як зберігаються дані.

Існують хмарні моделі, які дозволяють користувачам зберігати контроль над своїми даними. Хмарне зберігання розвивається в трьох напрямках, одне з яких допускає злиття двох інших для досягнення економічної ефективності й безпеки. Постачальники загальнодоступних хмарних систем зберігання даних надають інфраструктуру на умовах оренди (ресурси для довгострокового або короткострокового зберігання даних і смугу пропущення мережі). Приватні хмари використовують ті ж концепції, що й загальнодоступні, але в такій формі, що інфраструктура може бути надійно вбудована в приватну мережу користувача. Нарешті гібридні хмарні системи зберігання дозволяють з'єднати обидві моделі, визначаючи правила, що регулюють те, які дані необхідно зберегти в приватнім володінні, а які можна захистити в рамках публічних хмар (рисунок 2.6).

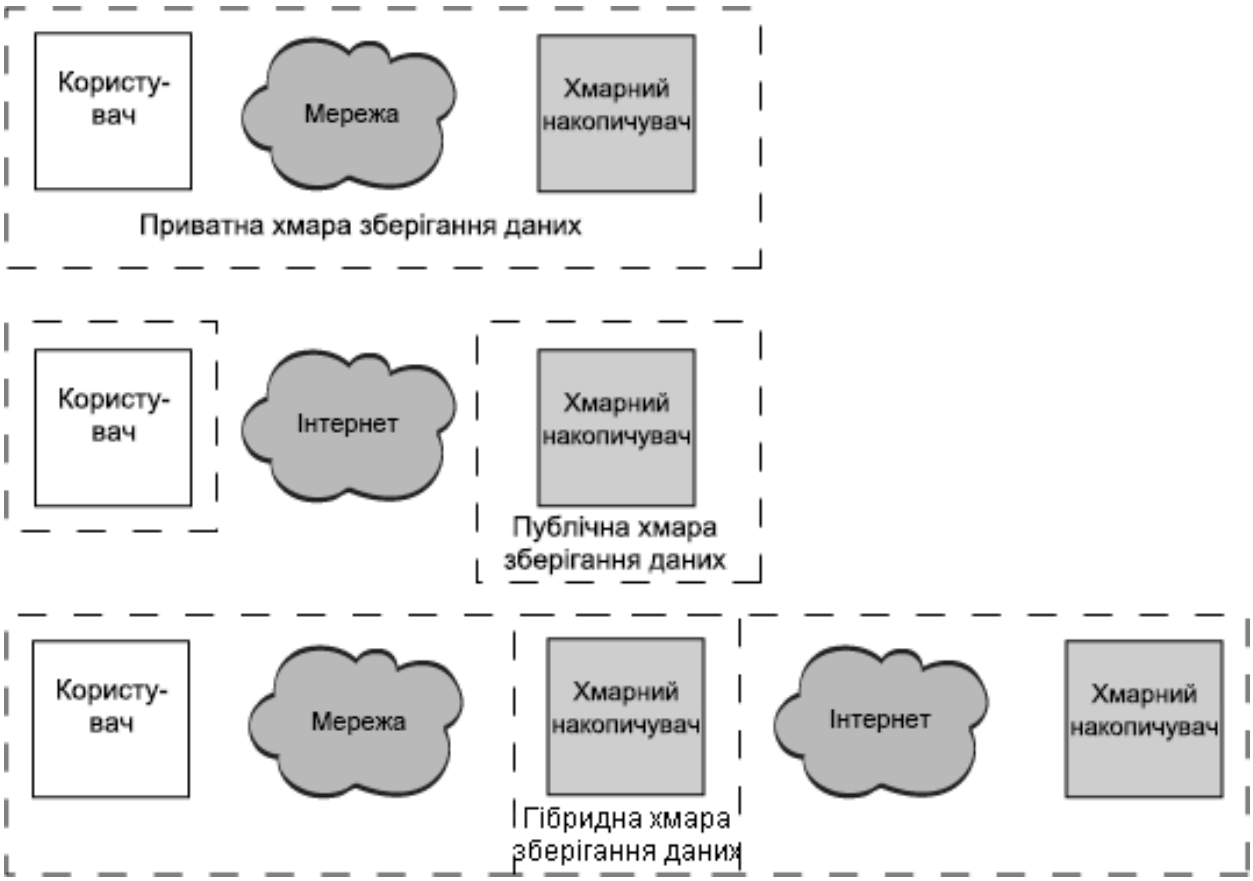

Рисунок 2.6 – Хмарні моделі зберігання даних

2.3 Аналіз алгоритмів оптимізації хмарного зберігання даних

Зберігання даних – уже не питання їх розміщення; у новому світі всюдисущих даних можливість знаходити, одержувати й ефективно використовувати дані вважається природною. Більша частина цих даних перебуває в хмарі, так що користувачеві потрібно знати, як оптимізувати зберігання даних і зробити так, щоб воно не стало слабкою ланкою в його хмарній платформі. Методи й програмні інструменти зберігання даних можуть допомогти в досягненні оптимізації даних і керуванні віртуалізованим зберіганням даних за допомогою програмного забезпечення.

Враховуючи величезна кількість даних, які щодня створюються в усьому

світі, не дивно, що компанії шукають більш ефективні й рентабельні засоби зберігання даних. З настанням ери хмарних обчислень миру знадобилося дуже багато дешевих обладнань зберігання даних, і вартість накопичувачів стала знижуватися. Однак дві нові тенденції – аналіз більших даних і «Інтернет речей» – загрожують звести нанівець економію від дешевих носіїв просто тому, що вимоги по масовій обробці даних стають приголомшуючими.

Аналіз більших даних несе із собою потреба в ефективному доступі до більших обсягів даних; він повинен бути не просто швидким, а доставляти ці дані в системи для їхньої візуалізації буквально в режимі реального часу.

А інтернет речей означає, що обсяг, що вводяться даних (і потреби в їхньому зберіганні) може вирости в 10 раз — і це ще консервативна оцінка.

Галузь приходить до того, що традиційна архітектура дискових масивів, у яку при необхідності можна додавати нові диски, уже не видасться достатньої, якщо у вас не саме дрібне підприємство.

Що означають ці тенденції для хмарних обчислень? Те, що левова частина послуг взаємодії, хостінгу додатків, того ж зберігання даних і мережної взаємодії з метою їх аналізу й компіляції буде відбуватися в хмарних системах.

Хмарне сховище даних корпоративного рівня побудоване на основі віртуалізованої інфраструктури з легкодоступними інтерфейсами й можливістю швидкого й надійного масштабування (в обох напрямках), здатної зберігати дані багатьох користувачів у тому самому місці, але повністю ізольовано й, у деяких випадках, з можливістю вимірювати обсяги даних, споживаних користувачами. Хмарне сховище й раніше застосовувалося для зберігання об'єктів (тобто воно здатне управляти даними як об'єктами), але тепер це поширюється на такі методи, як блокове зберігання (тобто керування даними як блоками в межах секторів і доріжок). Хмарне сховище повинне виглядати прозорим для користувачів і узгоджено управляти доступом за допомогою надмірності й розподілу ресурсів, а також версіями. Дані зберігаються в логічних пулах; фізичні ресурси зберігання об'єднані в пули, і із цих пулів фізичних запам'ятовувальних пристроїв створюється логічне сховище даних [21].

Віртуалізація систем зберігання даних полягає в наступному:

– віртуалізація блоків: відокремлює логічного сховище від фізичного, так що дані стають доступними, незалежно від фізичного обладнання;

– віртуалізація файлів: усуває залежність між даними, доступними на рівні файлів, і фізичним місцем зберігання цих файлів.

Можна виділити три основні методи оптимізації зберігання даних у хмарній системі: оптимізація даних, оптимізація бази даних і реалізація програмно-обумовленого сховища.

Оптимізація даних, імовірно, – найбільш значима з інновацій останнього часу в сфері програмного забезпечення, тому що вона дозволяє зберігати більше інформації в меншому фізичному просторі. Три інструменти оптимізації даних – дедуплікація, компресія й динамічне виділення ресурсів.

Дедуплікація (або виключення дублікатів) заощаджує простір, усуваючи дублюючі копії даних. У цьому процесі дані аналізуються на предмет виявлення й збереження унікальних фрагментів даних (байт-шаблонів). У міру аналізу даних інші фрагменти рівняються зі збереженою копією; у випадку збігу надлишкова версія заміняється посиланням на оригінал.

У типовім сховищі даних той самий байт-шаблон може втримуватися тисячі раз (залежно від розміру дедупліціруемих фрагментів). Чим менше розмір блоку зберігання, тем вище потенціал дедуплікації даних.

Стандартні інструменти стиску файлів, як правило, виявляють короткі ділянки рядків в одному файлі; при дедуплікації відшукуються більші розділи (або навіть цілі файли) у більших обсягах даних.

Ідеальними цілями дедуплікації, як правило, служать додатка для обміну файлами. Дедуплікацію також можна використовувати при передачі даних по мережі, щоб зменшити число переданих байтів.

Компресія (або інтелектуальне компресія) даних – знайомий термін, який означає просте кодування інформації зі зменшенням кількості бітів. Нагадаємо, що існує компресія двох типів – із втратам і без втрат. При компресії із втратами проводиться виявлення й видалення непотрібної інформації. При компресії без

втрат виявляються й віддаляються біти, необхідні для передачі даних (статистична надмірність), але не самі дані.

Звичайно компресії на рівні файлів зазнає більшість аудіофайлів і файлів зображень.

Механізм динамічного виділення ресурсів (або розріджених томів) лежить в основі концепції віртуалізації, але це не метод скорочення кількості даних; це модель розподілу ресурсів. Динамічне виділення ресурсів створює враження, що вам доступно більше фізичних ресурсів, чим є насправді. Воно дозволяє легко виділяти простір серверам за принципом достатності й оперативності. Цей механізм застосовується до систем з більшими обсягами дискової пам'яті, SAN і системам віртуалізації зберігання даних.

Шардинг теж можна розглядати як спосіб оптимізації даних, але це, скоріше, метод оптимізації бази даних. Шардинг – форма горизонтального масштабування даних – призначений для задоволення потреб у нарощуванні обсягів даних і доступу до даних; це процес зберігання записів даних в інших комп'ютерах, коли розмір бази даних обмежує продуктивність, тому що пропускна здатність по операціях читання й записи стає неприйнятною. Щоб упоратися з темпами росту обсягів даних, просто додаються нові машини.

При секціонуванні база даних ділиться на окремі логічні частини, що часто розподіляються між декількома серверами.

При вертикальнім секціонуванні стовпці даних розбиваються на окремі фрагменти й зберігаються в інших таблицях бази даних. Якоюсь мірою це робиться при нормалізації – організації інформації в таблицях реляційної бази даних для мінімізації надмірності даних. Прикладами вертикального секціонування служать зберігання рідке використовуваних або дуже широких стовпців на іншому обладнанні або відділення важких для пошуку динамічних даних від статичних даних, шукати які набагато легше.

При горизонтальнім секціонуванні (прикладом якого є шардинг) у різні таблиці містяться різні рядки. Наприклад, кожний рядок із записом, де в стовпці «Вік» значиться 50 або більш років, переходить в одну таблицю, а 49 або менш – в

іншу; таким чином, рядки розподіляються по двом таблицям — для тих, кому за 50, і для тих, хто молодше.

До недоліків шардинга ставиться необхідність у великій мері покладатися на межсоединения серверів, що веде до збільшення затримки, коли потрібно звертатися до декільком шардам (або коли дані або індекс сегментований тільки одним способом) і неможливість гарантувати погодженість шардів через відмінності в режимах обробки відмов серверів. Якщо шардинг бази даних виконується до її оптимізації, то керування стає занадто складним.

При використанні шардинга:

– можна чекати, що вам доведеться писати більш складні запити для пошуку даних;

– можливо, прийде мати справа з єдиною крапкою відмови, коли один шард, ушкоджений через проблему встаткування або мережі, може привести до відмови всієї таблиці;

– потрібно гарантувати, що сервери стійкого до відмов кластера містять копії шардов бази даних;

– необхідно координувати резервне копіювання всіх шардов;

– може збільшитися час додавання й видалення індексів і стовпців або зміни схеми бази даних [22].

2.4 Алгоритми міграції даних у хмарних сховищах

Традиційно оптимізація обчислювальних ресурсів у розподілених системах здійснюється за допомогою процедури балансування навантаження. Балансування полягає в призначенні виникаючих завдань певним обладнанням-оброблювачам, об'єднаним у кластер. Це має на увазі, що будь-яке завдання може бути оброблене кожним з обладнань у кластері. На жаль, ця умова не виконується в розподілених сховищах даних. Кожне з обладнань не здатне зберігати всі дані, які можуть бути

затребувані. Тому кожний елемент даних (блок даних, файл) зберігається лише на обмеженому наборі обладнань зберігання.

Одночасне переміщення більших обсягів даних приводить до різкого падіння продуктивності. Тому звичайно ухвалюють, що кожне з обладнань зберігання одночасно може брати участь не більш ніж в одній операції передачі даних. Також ухвалюють, що переміщувані елементи даних мають фіксований розмір і час передачі між будь-якими обладнаннями зберігання.

Завдання складання оптимального плану переміщення даних у сховищі називається завданням міграції даних. Отриманий у результаті її рішення план міграції даних дозволяє виконати процедуру міграції гранично швидко, тобто оптимізація плану міграції приводить до оптимізації часу міграції.

Доведене, що завдання міграції даних Np-Складна, т. е. неможливо одержати оптимальне рішення за поліноміальний час. Існують алгоритми апроксимації її рішення. Хмарні обчислення сильно міняють погляди на технології й, зокрема, на технології сховищ даних. Сховища мають фіксована кількість обладнань зберігання, з'єднаних мережею. Саме можливість масштабуватися відрізняє хмарні сховища від традиційних. Обладнання, що додаються до сховища або, що вивільняються зі сховища, будемо називати масштабованими обладнаннями. Додавання або видалення обладнань повинне відбуватися під час переконфігурації сховища.

Завдання міграції даних у загальному виді є Np-складною. Завдання міграції даних у масштабованім хмарнім сховищі є часткам случаємо завдання міграції. Перший критерій завдання – оптимізація часу міграції на масштабованих обладнаннях зберігання, тобто оптимізація часу масштабування. Другий критерій – оптимізація часу міграції на інших обладнаннях. Ефективне рішення поставленого завдання дозволить гранично швидко масштабувати сховища даних, знижуючи вартість на оренду обладнань зберігання [23].

Перед переконфігурацією сховища необхідно обчислити оптимальне розташування даних у сховище. Вихідними даними цього завдання є відображення, у якім кожному елементу даних зіставляються обладнання

зберігання, на яких він повинен бути після переконфігурації. Завдання є Np-Складної, але були розроблені поліноміальні апроксимаційні алгоритми її рішення. Обчисливши нове розташування даних і маючи старе розташування, можна побудувати спрямований мультиграф без циклів G, що демонструє переміщення елементів даних зі старої конфігурації в нову.

Цей граф називають графом вимог:

$$\begin{gathered} G = (V, E, P) \\ E \in V \times V \\ P : E \to IN \\ v, w \in V, P(v, w) = 0 \text{ якщо } (v, w) \notin E \end{gathered} \tag{2.1}$$

де V – обладнання зберігання;

E – операція переміщення;

P – функція ваг мультиграфу.

План міграції можна розбити на кроки, оскільки всі елементи даних мають фіксований розмір і однаковий час передачі. Завдання міграції даних полягає в складанні плану переміщення даних між обладнаннями зберігання згідно із графом вимог за мінімальне число кроків. Очевидно, що завдання в такому виді елементарно зводиться до завдання розфарбування дуг мультиграфа. Мінімальне число квітів розфарбування дуг мультиграфа називають хроматичним індексом мультиграфа й позначають χ. Слід помітити, що напрямок передачі даних у сховище не має значення з погляду завдання міграції. Обладнання вважається зайнятим незалежно від того, ухвалює воно дані або передає. Заміна спрямованого мультиграфа на ненаправлений дозволить скоротити мінімальна кількість квітів розфарбування:

$$\begin{gathered} G = (V, E, P) \\ E \in V \times V \\ P : E \to IN \\ v, w \in V \to P(v, w) = P(w, v) \end{gathered} \tag{2.2}$$

2.5 Алгоритм балансування навантаження в хмарнім сховищі

Відмітною характеристикою хмарних сховищ є реконфігуруємость їх структури залежно від споживаних ресурсів. Це у свою чергу дозволяє впроваджувати алгоритми оптимізації в плані розміщення даних усередині дискового простору, а також управляти зміною кількості використовуваних системою обладнань. При цьому процес оптимізації розміщення не повинен приводити до зниження якості обслуговування клієнтів СХД, для чого в алгоритмах необхідно враховувати пропускну здатність мережі й максимальний обсяг даних, який можна передавати в один момент часу. Крім того необхідно враховувати поточне завантаження самих обладнань, а також їх розташування відносно один одного й клієнтів, що підключаються до них.

Для оптимізації механізмів доступу до даних побудуємо загальну модель доступу до даних системи зберігання.

Нехай R = (U,M,Q), де U = {u1,u2,…} – множина користувачів; M = {m1,m2,…} – множина унікальних елементів даних, розташованих на обладнаннях зберігання.

При цьому мінімальною одиницею даних m_i вважається файл, що має обов'язкову властивість *h* – розмір.

Тоді функція розподілу елементів даних по обладнаннях зберігання ухвалює вид P:Mc → D. Виходячи з викладеного вище, запишемо вимогу користувача до елементів даних Q.

$$Q:U \rightarrow X \quad Mc,$$

де X – множина даних запитаних безліччю користувачів U.

Тоді сховище даних можна записати у вигляді кортежу

$$S=(Mc,D,P,L,C,R,G),$$

де D={d1,d2,…} – множина обладнань зберігання;

L={l1,l2,…} – множина значень, що характеризує завантаження кожного обладнання зберігання (кількість одночасних звернень користувачів до

конкретного обладнання);

C={c1,c2,…} – множина значень, що характеризує обсяг кожного з обладнань у сховище;

G→N –коефіцієнт, що характеризує географічний (топологічний) пріоритет використання сховища.

Як правило, для великих хмарних структур використовуються консолідовані сховища, що полягають із ферм, що поєднують кілька сховищ у єдиний масив. Представимо його як Sfarm={S1,S2,…}. Тому що характеристики вимог користувачів міняються в часі, перетворимо кортеж вимог R(t)=(U,Mc,Q(t)). Тоді $Q(t):U \to X$ Mc – вимоги користувача до елементів даних, що міняються в часі. Тому що крім активності користувача змінюються властивості сховища, запишемо кортеж сховища залежно від часу S(t).

S(t)=(Mc(t),D(t),P(t),L(t),C,R(t),G), де D(t)={d1,d2,…} безліч обладнань зберігання, що міняються в часі, таких що t, D(t)>0;

$P(t):Mc \to D$ – функція розподілу елементів даних по обладнання зберігання, що міняється в часі. При цьому для оптимізації витрат на апаратні ресурси й скорочення одночасно використовуваних обладнань уведемо кортеж відносин Scloud(t).

Scloud(t)={S(t),D(t),Duse(t)}, де $\forall$ t, Duse(t)$\in$D(t) безліч обладнань зберігання використовуваних у масштабованім сховищі S у момент часу t. Крім того, при масштабуванні сховища й міграції даних повинне виконуватися умова $\forall$ t, i, j $i \neq j \Rightarrow di(t) \cap Dj(t)=0$, тобто при міграції даних сховища не повинні використовувати ті самі обладнання. Це дозволить як гарантувати швидкість обробки інформації, так і забезпечити прийнятний час реконфігурації [24].

Таким чином, для мінімізації кількості одночасно використовуваних обладнань зберігання в рамках одного масштабованого сховища й максимізації кількості оброблених запитів користувачів в одиницю часу, уведемо цільову функцію виду:

$$\sum_{i=1}^{N} Pi(t) \to \min$$
$$\sum_{i=1}^{N} LiPi(t)Ri(t) \to \max \quad (2.3)$$

де N – загальна кількість заявок, що зробили в систему на інтервалі часу ΔT.

Масштабованість можна розглядати з декількох точок зору, але нас в основному цікавить виділення хмарних ресурсів зберігання на вимогу. Можливість нарощувати ресурси зберігання (як нагору, так і вниз) означає поліпшену економічну ефективність для користувача й підвищену складність для постачальника хмарних послуг.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

3.1 Модель і розбивка масштабованого хмарного сховища

Опишемо модель масштабованого хмарного сховища на основі сховища без обмежень обсягу обладнань зберігання. Щоб урахувати особливості масштабованого хмарного сховища, явно виділимо підмножину масштабуючих обладнань зберігання, яке необхідно вивести з ладу (або додати) у результаті пере конфігурації [25].

Такою, що масштабує (scaling) підмножиною S називають підмножину вершин, які необхідно вивести з ладу або додати до сховища в результаті переконфігурації. Серед операцій передачі даних не може бути операцій передачі з один масштабуючого обладнання на інше, тобто

$$\forall v, w \in S \rightarrow V \Rightarrow P(v, w) = 0 \tag{3.1}$$

Масштабованим хмарним сховищем G називають сховище з явно виділеним масштабуючим підмножиною S.

$$\begin{gathered} G = (V, E, P, S) \\ \forall v, w \in S \rightarrow V \Rightarrow P(v, w) = 0 \end{gathered} \tag{3.2}$$

Під час масштабування необхідно виконати процедуру міграції на всіх масштабуючих обладнаннях S. Після цього їх можна визволити й виконати міграцію даних на обладнаннях, що залишилися, зберігання.

Розділимо процедуру міграції на дві частини: масштабування й залишкову міграцію. Для рішення подібних завдань часто використовують методи поділу графів на підграфи. Розділимо граф масштабованого хмарного сховища G на два підграфа: масштабуючий підграф Gs і залишковий Gr.

Масштабуючим подграфом Gs будемо називати підграф масштабованого хмарного сховища G, утворений безліччю вершин S і безліччю вершин Sadj, суміжними з S, а також усіма дугами, що з'єднують вершини S з Sadj.

$$\begin{gathered} Gs = (Vs, Es, P) \\ Vs \in V \\ Es \in E \\ v \in sadj \rightarrow V \Leftrightarrow \exists w \in S, P(v, w) > 0 \\ Vs = S \cup Sadj \\ (v,w) \in ES \Leftrightarrow v \in S, w \in Sadj, P(v, w) > 0 \end{gathered} \tag{3.3}$$

В останній формулі визначення вираження $v \in S$, $w \in Sadj$, $P(v, w) > 0$ еквівалентно вираженню $v \in Sadj$, $w \in S$, $P(v, w) > 0$, оскільки граф G ненаправлений. У лівій частині рис. 3.1 показаний граф G, підмножина S (чорні вершини) і Sadj (заштриховані вершини).

Залишковим (residual) підграфом GR будемо називати підграф масштабованого хмарного сховища G, утворений усіма дугами графа G без масштабирующего підмножини S, і всі дуги, що з'єднують ці вершини.

$$\begin{gathered} Gr = (Vr, Er, P) \\ Vr \in V \\ Er \in E \\ Vr = V \setminus S \\ (v,w) \in ER \Leftrightarrow v, w \in Vr, P(v, w) > 0 \end{gathered} \tag{3.4}$$

Очевидно, що множина вершин Sadj належить як Gs, таки Gr. Дуги, що з'єднують вершини Sadj, лежать тільки в Gr. У правій частині рисунку 3.1 зображений граф, розділений на два подграфа Gs і Gr.[26]

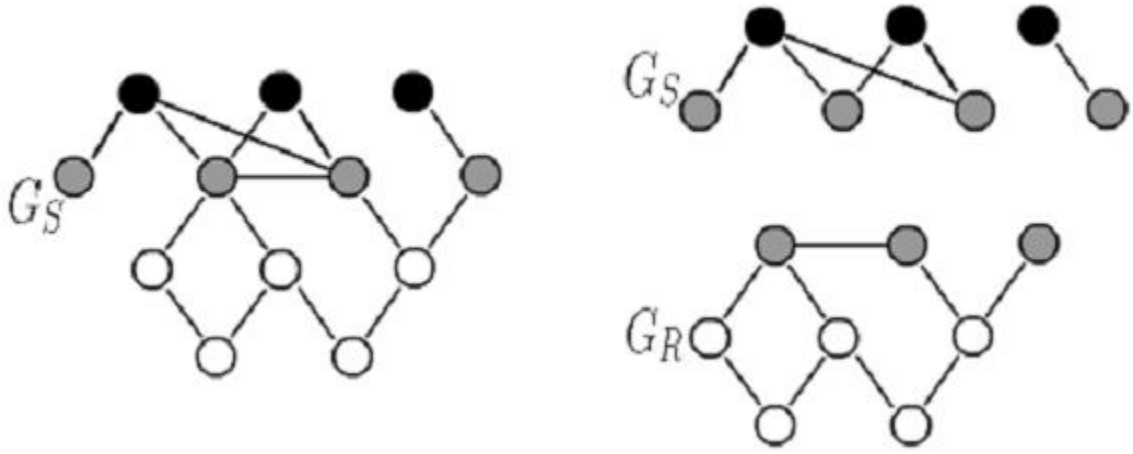

Рисунок 3.1 – Розбивка графа масштабованого хмарного сховища G на два підграфу: Gs і Gr

3.2 Алгоритми міграції даних у масштабованому хмарному сховищі

Описано алгоритм міграції даних у масштабованім хмарнім сховищі оптимальний результат, що дає, за першим критерієм – оптимальності часу масштабування. Для цього доведемо, що масштабирующий подграф Gs є двочастковим мультиграфом, для якого існують оптимальні поліноміальні алгоритми розфарбування дуг.

Масштабирующий подграф Gs є двочастковим. Дійсно, масштабуючий подграф складається з безлічі вершин S і вершин Sadj, суміжних з S. Доведемо, що безлічі вершин S і Sadj розбивають граф Gs на дві частки, тобто в графові немає дуг, що йдуть із S в S або ж з Sadj в Sadj.

Вершини з безлічі Sadj не мають загальних дуг, оскільки це суперечить визначенню масштабуючої підмножини, що входить до складу масштабованого хмарного сховища. Також будь-яка його дуга з'єднує вершини з S тільки з вершинами з Sadj. Отже, вершини з Sadj також не можуть мати загальних дуг. Таким чином, безлічі вершин S і Sadj становлять дві частки двочасткового мультиграфа Gs.

Алгоритм 1 (Міграція даних). Для рішення завдання будемо використовувати метод поступків. Виділивши два критерії оптимізації, будемо оптимізувати алгоритм за першим критерієм, для чого буде потрібно поступитися ефективністю за другим критерієм. Перший приватний критерій – час масштабування, тобто міграція даних на масштабуючему підграфе GS. Другий критерій – час міграції на інших обладнаннях, тобто міграція даних на підграфе Gr. Опишемо алгоритм у вигляді послідовності кроків, описаних вище.

Крок 1. Виділити підграф Gs із графа G на основі відомої підмножини S.

Крок 2. Виділити підграф Gr з G на основі підграфа Gs.

Крок 3. Використовувати алгоритм розфарбування ребер двочасткового мультиграфа для розрахунку плану міграції двочасткового мультиграфа Gs.

Крок 4. Використовувати переборний алгоритм розфарбування ребер мальтиграфа для розрахунку плану міграції мультиграфа Gr.

Крок 5. Одержати загальний плану міграції масштабованого хмарного сховища G шляхом послідовного об'єднання планів міграції мультиграфа Gs із планом міграції Gr.

Алгоритм 2 (Поліноміальна міграція даних). Кроки аналогічні алгоритму 1. Але на кроці 4 використовується поліноміальний алгоритм розфарбування ребер мультиграфа замість переборного алгоритму.

Запропоновані алгоритми 1 і 2 оптимальні за першим критерієм завдання міграції даних у масштабованім хмарнім сховищі. Розглянемо спочатку алгоритм 1. Відповідно до доведеної леми 1, подграф Gs є двочастковим мультиграфом, і для знаходження оптимального розфарбування дуг Gs в алгоритмі використовується поліноміальний алгоритм, який дає оптимальний час міграції підграфа Gs. Виходячи з постановки завдання, оптимальність алгоритму за першим критерієм забезпечується завдяки оптимальності часу міграції підграфа Gs (крок 3 алгоритму). Тобто алгоритм 1 є оптимальним за першим критерієм. Поліноміальний алгоритм 2 відрізняється від 1 тільки кроком 4, а оптимальність, як показано вище, забезпечується на кроці 3. Отже, алгоритм 2 також є оптимальним за першим критерієм.

Запропонований алгоритм 2 має поліноміальну обчислювальну складність. Кроки 1 і 2 алгоритму 2 є поліноміальними, оскільки обчислення вершин Sadj, суміжних з S, – завдання тривіальне, а виділення підграфів Gs і Gr, що полягають із заданих вершин, має лінійну складність від кількості вершин O(|V|). Кроки 3 і 4 також є поліноміальними, їх обчислювальна складність рівна O(|a| log Δ) і O(|A|(|V | + δ)), відповідно до алгоритмів розфарбування ребер, де A – множина ребер; V – множина вершин; Δ – максимальний ступінь вершин; δ – деяка константа.

Результат виконання кроків 3 і 4 – два плани міграції переміщення, що полягають із послідовності операцій, даних (дуг підграфів), а крок 5 складається із простої операції послідовного об'єднання двох планів міграції. Очевидно, що крок 5 має лінійний час виконання O(|A|), що залежить від числа ребер графа G.

Оскільки алгоритм 2 є послідовним об'єднанням кроків 1-5, його обчислювальна складність рівна складності найбільш витратного кроку: max{O(|V |), O(|a| log Δ), O(|A|(|V | + δ)), O(|A|)} = max{O(|a| log Δ), O(|A|(|V | + δ))}. Із цього випливає, що алгоритм має поліноміальну складність [27].

3.3 Завдання і алгоритм оцінки ефективності

Використовуючи традиційні алгоритми міграції даних, масштабування хмарного сховища можна зробити тільки після повного виконання процедури міграції даних. Це пов’язане з тим, що масштабуючіеся обладнання зберігання можуть бути задіяні в процедурі міграції даних до самих останніх кроків. Розроблений алгоритм виділяє в графові сховища G масштабуючий подграф Gs, що містить усі масштабующеєся обладнання. Це дозволяє скоріше виконати процедуру міграції на цих обладнаннях і визволити їх з метою економії витрат на оренду обладнань. Зворотною стороною алгоритму є необхідність виконання залишкової міграції в подграфе Gr після виконання міграції в Gs і послідовне об'єднання кроків міграції, яке збільшує загальний час міграції.

Із практичної точки зору цікаві відповіді на наступні питання:

– на скільки зменшується час (кількість кроків) масштабування;

– на скільки збільшується час повної процедури міграції.

Такі оцінки зручно робити у відсотках щодо загального часу міграції G з використанням традиційних алгоритмів. Кількість кроків у плані міграції дорівнює кількості квітів розфарбування дуг відповідного мультиграфа сховища, тобто його хроматичному індексу $\chi 0$. Значення хроматичних індексів $\chi 0$ (G), $\chi 0$(Gs) і $\chi 0$(Gr) відповідають кількості кроків у плані міграції для відповідних графів сховищ. У якості наближеної оцінки $\chi 0$ можна використовувати максимальний ступінь вершин мультиграфа Δ плюс 1. Будемо позначати цю оцінку як $\Delta 0$. Використання наближених оцінок $\Delta 0$ замість $\chi 0$ дозволяє швидко й досить точно зробити оцінку, не прибігаючи до трудомісткої процедури обчислення розфарбування ребер мультиграфа.

Ефективність переборного алгоритму 1 можна оцінити формально, оскільки він дає оптимальний результат. Ефективність алгоритму 2 оцінювати не будемо, оскільки він є наближенням алгоритму 1 і відрізняється від алгоритму 1 лише планом міграції в підграфі Gr, що не суттєво з погляду першого критерію оптимізації.

Виходячи із завдання оцінки ефективності алгоритмів міграції даних у хмарнім сховищі потрібно використовувати наступні параметри оцінки:

$Pstep = \chi 0(G) - \chi 0(Gs) \approx \Delta 0(G) - \Delta 0(Gs)$ – час (кіл-у кроків) масштабування, який був зекономлене врезультате застосування алгоритму;

$Prel = Pstep/\chi 0(G) \approx (\Delta 0(G) - \Delta 0(Gs))/\Delta 0(G)$ – відносний виграш від масштабування, відповідний до зекономленого часу в порівнянні із традиційним алгоритмом;

$Lstep = \chi 0(Gs) + \chi 0(Gr) \approx \Delta 0(Gs) + \Delta 0(Gr)$ – загальний час міграції даних з новим алгоритмом;

$Lrel = Lstep/\chi 0(G) - 1 \approx (\Delta 0(Gs) + \Delta 0(Gr))/\Delta 0(G) - 1$ – відносний програш часу від поділу графа, що відповідає додатково витраченому часу на міграцію від загального часу міграції даних.

Можна обчислити абсолютний виграш від масштабування в машино-годинах (точніше в машино-кроках), які будуть зекономлені в результаті застосування алгоритму: P = Pstep * |S|.

Програш часу процедури міграції (абсолютний програш L) не приводить до втрати ресурсів, але приводить до збільшення часу роботи сховища в режимі переконфігурації й, відповідно, до збільшення часу між послідовними процедурами міграції.

Запропоновано два алгоритми рішення завдання: переборний алгоритм і апроксімаційний алгоритм поліноміальної обчислювальної складності max{O(|a| log Δ), O(|A|(|V | + δ))}, де A – безліч ребер; V – безліч вершин; Δ – максимальний ступінь вершин; δ – деяка константа. Алгоритми дають оптимальний результат за критерієм мінімізації часу масштабування. Оптимальна швидкість масштабування дозволить хмарним сховищам гранично швидко орендувати й вивільняти обладнання зберігання даних.

4 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМНОЇ СИСТЕМИ

4.1 Інтеграція зі сторонніми API

Успішні сайти все частіше не повністю автономні. Для того щоб зацікавити існуючих і знайти нових користувачів, інтеграція із соціальними мережами обов'язкова. Для вказівки місцезнаходження магазинів і місць надання інших послуг дуже велике значення має використання сервісів геолокації й відображення на карті. І на цьому не зупиняються: усе більше організацій усвідомлюють, що надання API допомагає розширити перелік послуг і зробити їх більш корисними.

Соціальні медіа — це відмінний спосіб просунути ваш продукт або сервіс: можливість для користувачів ділитися вашим контентом у соціальні медіа дуже важлива. Сьогодні, що коли домінують сервіси соціальних мереж — Facebook і Twitter, Google+ може посилено боротися за свою частку, але не випливає зовсім не брати його в розрахунок: він працює за підтримки однієї з найбільших інтернет-компаній у світі. У сайтів Pinterest, Instagram є своє місце, але в них, як правило, невелика й більш специфічна аудиторія (наприклад, якщо сайт пов'язаний зі створенням виробів способом «зроби сам», те користувач напевно захоче підтримувати Pinterest). Myspace ще повернеться. Варто відзначити, він зроблений на Node.

Плагіни соціальних медіа й продуктивність сайту - більша частина інтеграції соціальних медіа — це справа клієнтської частини. Відбувається посилання на відповідні файли Javascript на вашій сторінці, і це включає як вхідний контент (наприклад, три верхні пости з вашої сторінки на Facebook), що так і виходить (наприклад, можна зробити твіт про сторінку, на якій ви перебуваєте). Найчастіше це простий спосіб інтеграції соціальних медіа, але в нього теж є своя ціна: я бачив, як сторінки завантажуються вдвічі, а то й утроє довше через додаткові --Запитів. Якщо продуктивність сторінки важлива (а вона

повинна бути важлива, особливо якщо розроблювач орієнтується на мобільних користувачів), потрібно ретельно продумати, як інтегрувати соціальні медіа.

При цьому код, який дозволяє включити кнопку Facebook «Подобається» або кнопку «Tweet», використовує браузерні cookie, щоб відправити повідомлення від імені користувача. Переміщення цього функціонала на серверну сторону буде непростим, а в деяких випадках і зовсім неможливим. Так що, якщо вам потрібний саме цей функціонал, найкращим рішенням може бути підключення відповідної сторонньої бібліотеки, навіть незважаючи на те, що це може вплинути на продуктивність сторінки. Урятувати ситуацію може те, що Api-Інтерфейси Facebook і Twitter досить поширені: дуже велика ймовірність, що браузер користувача їх уже закешировал, а в цьому випадку вони вплинуть на продуктивність незначно.

Пошук твітів. Припустимо, ми прагнемо вказати десять останніх твітів, які містять хеш-тег `#meadowlarktravel`. Ми могли б використовувати для цього компонентів клієнтської частини, але він буде включати додаткові запити HTTP. Більше того, якщо ми робимо це на серверній стороні, у нас з'являється можливість кэширования твитов для підвищення продуктивності. У цьому випадку ми також можемо відправляти в чорний список твіти недоброзичливців, тоді як на стороні клієнта це зробити було б суттєво складніше.

Twitter, як і Facebook, дозволяє вам створити додатка. Це небагато невідповідна назва: додаток Twitter нічого не робить (у традиційному змісті). Це скоріше набір облікових даних, які ви можете використовувати для створення реального додатка на вашому сайті. Найпростіший і найбільш універсальний спосіб одержання доступу на Twitter API — створити додаток і використовувати його для одержання токена доступу.

Клацніть на значку в лівому верхньому куті, потім виберіть Мої додатки. Натисніть «Створити новий додаток» і додержуйтеся інструкцій. Коли з'явиться додаток, ви побачите, що у вас тепер є споживчий ключ і споживчий секрет. Споживачский секрет, на що вказує назва, повинен зберігати секрет: ніколи не включайте його у відповіді, що відправляються на сторону клієнта. Якби хтось

ззовні одержав доступ до цього секрету, він міг би створювати запити від імені вашого додатка, що могло б мати сумні наслідки, якщо їх використання шкідливе.

Тепер у нас є споживчий ключ і споживчий секрет і ми можемо спілкуватися з Twitter REST API.

Щоб зберігати код в акуратному виді, помістимо код Twitter у модуль, названий lib/twitter.js:

```
var https = require('https'); module.exports = function(twitteroptions){
return {
search: function(query, count, cb){
// те, що потрібно зробити
}
};
};
```

Цей шаблон напевно починає видатися вам знайомим. Наш модуль експортує функцію, у яку зухвала сторона передає об'єкт конфігурації. Вертається об'єкт, що містить методи. Таким чином, ми можемо додати функціонал у наш модуль. Метод search та використання бібліотеки:

```
var twitter = require('./lib/twitter')({
consumerkey:        credentials.twitter.consumerkey,        consumersecret:
credentials.twitter.consumersecret,
});

twitter.search('#meadowlarktravel', 10, function(result){
// твіти будуть в result.statuses

});
```

Перш ніж реалізувати метод `search`, буде потрібно надати певну функціональність для аутентифікації користувача в Twiter. Процес простий: використовано HTTPS для запиту токену доступу, що базується на власному споживчому ключі й споживчому секреті. Потрібно зробити це тільки раз: токени в Twitter даються безстроково (втім, можете анулювати їх вручну). Оскільки ми не прагнемо запитувати токен доступу щораз, його закешує для подальшого використання.

Спосіб, за допомогою якого ми побудували модуль, дозволяє створити приватяну функціональність, яка буде недоступна зухвалій стороні.

Зокрема, єдина функція, яка доступна зухвалій стороні, — це module.exports. Оскільки ми повертаємо функцію стороні, що викликає, буде

доступна тільки вона. Виклик функції дає в результаті об'єкт, і тільки властивості цього об'єкта доступні зухвалій стороні. Отже, ми збираємося створити змінну accesstoken, яку будемо використовувати для кешування нашого токена доступу, і функцію getaccesstoken, яка цей токен одержить. При першому за- пуску вона створить запит Twitter API для одержання токена доступу. Наступні виклики просто повертають значення accesstoken:

```
     var https = require('https'); module.exports = function(twitteroptions){
     // ця змінна буде невидимої поза цим модулем var accesstoken;

     // ця функція буде невидимої за межами цього модуля function getaccesstoken(cb){
     if(accesstoken) return cb(accesstoken);
     // те, що потрібно зробити: одержання токена доступу
     }

     return {
     search: function(query, count, cb){
     // те, що потрібно зробити
     },
     };
     };
```

Оскільки getaccesstoken може вимагати асинхронний виклик до Twitter API, ми повинні надати зворотний виклик, який буде здійснюватися, коли значення accesstoken дійсно. Тепер, коли ми встановили базову структурові, реалізуємо getaccesstoken:

```
 function getaccesstoken(cb){ if(accesstoken) return cb(accesstoken);

 var bearertoken = Buffer( encodeuricomponent(twitteroptions.consumerkey) + ':' + encodeuricomponent(twitteroptions.consumersecret)
 ).tostring('base64');

 var options = {
 hostname: 'api.twitter.com', port: 443,
 method: 'POST',


 path: '/oauth2/token?grant_type=client_credentials', headers: {
 'Authorization': 'Basic ' + bearertoken,
 },
 };

 https.request(options, function(res){ var data = '';
 res.on('data', function(chunk){ data += chunk;
 });
 res.on('end', function(){
 var auth = JSON.parse(data); if(auth.token_type!=='bearer') {
```

```
console.log('Twitter auth failed.'); return;


});
}
accesstoken = auth.access_token; cb(accesstoken);}).end();
}
```

Подробиці побудови цього виклику доступні на сторінці документації для розроблювачів Twitter для аутентифікації «тільки для додатків». У принципі, потрібно зробити токен пред'явника (bearer token), що буде base 64-кодованою комбінацією споживчого ключа й споживчого секрету. Після того як ми сконструюємо токен, можемо викликати /oauth2/token API із заголовком Authorization, що містять токен пред'явника для запиту токена доступу. Зверніть увагу на те, що ми повинні використовувати HTTPS: якщо ви спробуєте зробити цей виклик за допомогою HTTP, те передасте свій секрет у незашифрованому виді, а API попросту відключить вас.

Після того як одержимо повну відповідь від API (ми очікуємо подію end потоку відповіді), ми можемо розібрати JSON, переконатися, що тип токена bearer, і далі йти своєю дорогою. Проводиться кэшування токену доступу, потім виклик функцію зворотного виклику.

Тепер, раз у нас є механізм одержання токена доступу, ми можемо робити виклики API.

Реалізуємо наш метод пошуку:

```
search: function(query, count, cb){ getaccesstoken(function(accesstoken){
var options = {
hostname: 'api.twitter.com',

port: 443, method: 'GET',
path:   '/1.1/search/tweets.json?q='   +   encodeuricomponent(query)   +
'&count=' + (count || 10),
headers: {
'Authorization': 'Bearer ' + accesstoken,
},
};
https.request(options, function(res){ var data = '';
res.on('data', function(chunk){ data += chunk;
});
res.on('end', function(){ cb(JSON.parse(data));
});
},
});
}).end();
```

Тепер є можливість шукати твіти — тому що ми будемо відображати їх на нашому сайті? По великому рахункові, це залишається на ваш розсуд, але є кілька речей, які ми повинні розглянути. Twitter зацікавлений у тому, щоб його дані використовували у відповідному стилі. Для цього в нього є вимоги відображення, що використовують функціональні елементи, які ви повинні включити для відображення в Twitter.

У вимогах можливі певні маневри (наприклад, якщо ви відображаєте твіти на обладнанні, яке не підтримує зображення, немає потреби додавати аватар), але по більшій частині кінцевий підсумок повинен бути чимсь дуже схожим на те, як виглядає вбудований твіт. Це припускає багато роботи, і є спосіб її зробити... Але він містить у собі посилання на бібліотеки віджетів Twitter, а це і є той самий Http-Запит, якого ми намагаємося уникнути.

Якщо потрібно відображати твіти, найкраще використовувати бібліотеку виджетов Twitter, незважаючи на те що це додасть додатковий Http-запит (знову ж у силу Twitter цей ресурс напевно закешований браузером, так що зниження продуктивності може виявитися незначним). Для більш складного використання API вам потрібний доступ до REST API із серверної сторони, так що ви в підсумку напевно будете використовувати REST API у зв'язуванні зі скріптами у веб-інтерфейсі.

4.2 Загальна структура

Java – об'єктно-орієнтована мова програмування, розроблювальний компанією Sun Microsystems з 1991 року й офіційно випущений 23 травня 1995 року. Споконвічно нова мова програмування називалася Oak (James Gosling) і розроблявся для побутової електроніки, але згодом був перейменований в Java і став використовуватися для написання аплетів, додатків і серверного програмного забезпечення

Відмінною рисою Java у порівнянні з іншими мовами програмування загального призначення є забезпечення високої продуктивності програмування, ніж продуктивність роботи додатка або ефективність використання їм пам'яті.

В Java використовуються практично ідентичні угоди для оголошення змінних, передачі параметрів, операторів і для керування потоком виконанням коду. В Java додані всі гарні риси C++.

Java надає для широкого використання свої апплети (applets) — невеликі, надійні, динамічн мережні додатки, що не залежать від платформи активні, що вбудовуються в сторінки Web. Аплети Java можуть настроюватися й поширюватися споживачам з такою ж легкістю, як будь-які документи HTML.

Java вивільняє міць об'єктно-орієнтованої розробки додатків, поєднуючи простий і знайомий синтаксис із надійному й зручним у роботі середовищем розробки. Це дозволяє широкому колу програмістів швидко створювати нові програми й нові аплети

Java надає програмістові багатий набір класів об'єктів для ясного абстрагування багатьох системних функцій, використовуваних при роботі з вікнами, мережею й для вводу-висновку. Ключова риса цих класів полягає в тому, що вони забезпечують створення незалежних від використовуваної платформи абстракцій для широкого спектра системних інтерфейсів.

Програми на Java транслюються в байт-код, виконуваний віртуальною машиною Java (JVM) - програмою, що обробляє байтовий код і передавальної інструкції встаткуванню як інтерпретатор. Трансляція в байтовий код збільшує швидкість виконання й зменшує розмір Java програм.

Гідність подібного способу виконання програм - у повній незалежності байт-коду від операційної системи й устаткування, що дозволяє виконувати Java-Додатка на будь-якому обладнанні, для якого існує відповідна віртуальна машина. Іншої важливою особливістю технології Java є гнучка система безпеки завдяки тому, що виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують установлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим

комп'ютером) викликають негайне переривання.

Величезна перевага Java полягає в тому, що на цій мові можна створювати додатка, здатні працювати на різних платформах. До мережі Internet підключені комп'ютери самих різних типів –PC, Macintosh, робочі станції Sun і так далі. Навіть у рамках комп'ютерів, створених на базі процесорів Intel, існує кілька платформ, наприклад, різні різновиди операційної системи UNIX із графічною оболонкою. Тим часом, створюючи сервер Web у мережі Internet, хотілося б, щоб їм могло користуватися як можна більше число людей. У цьому випадку виручать додатка Java, призначені для роботи на різних платформах і не залежні від конкретного типу процесора й операційної системи.

Основні можливості мови Java:

– автоматичне керування пам'яттю;

– розширені можливості обробки виняткових ситуацій;

– багатий набір засобів фільтрації введення/висновку;

– набір стандартних колекцій, таких як масив, список, стік і т.п.;

– наявність простих засобів створення мережних додатків (у тому числі з використанням протоколу RMI);

– наявність класів, що дозволяють виконувати Http-Запити й обробляти відповіді;

– вбудовані в мову засобу створення багатопоточних додатків;

– уніфікований доступ до баз даних:

– на рівні окремих Sql-запитів – на основі JDBC, SQLJ;

– на рівні концепції об'єктів, що володіють здатністю до зберігання в базі даних - на основі Java Data Objects і Java Persistence API;

Мова Java є об'єктно-орієнтованим і поставляється з досить об'ємною бібліотекою класів. Так само як і бібліотеки класів систем розробки додатків мовою C++, бібліотеки класів Java значно спрощують розробку додатків, представляючи в розпорядження програміста потужні засоби рішення розповсюджених завдань. Тому програміст може більше уваги приділити рішенню прикладних завдань

Мова Java спеціально орієнтований на самі передові технології, пов'язані з мережею Internet. Зростаюча популярність Internet і, особливо, серверів Web, створює для програмістів нові можливості для реалізації своїх здатностей.

Як приклад, використовуються два сервіси, що надають послуги по хмарнім сховищу файлів: Dropbox і Google Drive. Обоє ці сервісу мають публічне API, яке дозволяє працювати з ними.

Дозволяє API підвищення безпеки: не буде потрібно вводити свої персональні дані (логін і пароль) у сторонні додатки. Завдяки API буде потрібно лише підтвердити використання акаунта Dropbox (через мобільне додатки або через веб-інтерфейс).

Папки для додатків: додавання підтримки додатків, які можуть тільки читати або зберігати в 1 папці в Dropbox акаунте. Можна перейменувати або перемістити цю папку додатка куди завгодно у своєму акаунте, і додаток буде продовжувати працювати як звичайно.

Більше підтримуваних платформ: API був розроблений для мобільних додатків. Однак додана поліпшена підтримка веб-додатків. Вона містить у собі нові уроки, документації й пакети SDK для Python, Ruby і Java-розробників.

4.3 Опис проведеного тестування

На рисунку 4.1 зображене головне вікно програми після її завантаження. В інтерфейсі відображаються файли, які доступні на хмарних сховищах Dropbox і Google Drive, а так само відповідні дії, які можна виконати із цими файлами.

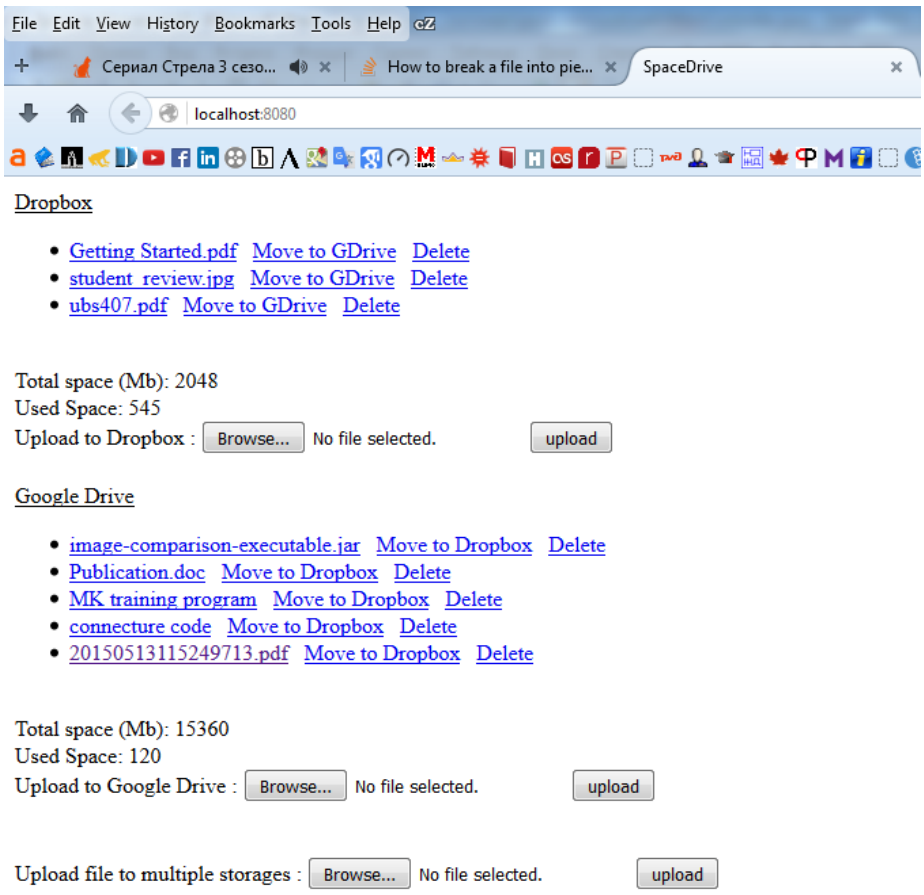

Рисунок 4.1 – Головне вікно програми

На малюнках 4.2 і 4.3 зображені файли, які перебувають на кожнім хмарнім сховищі окремо.

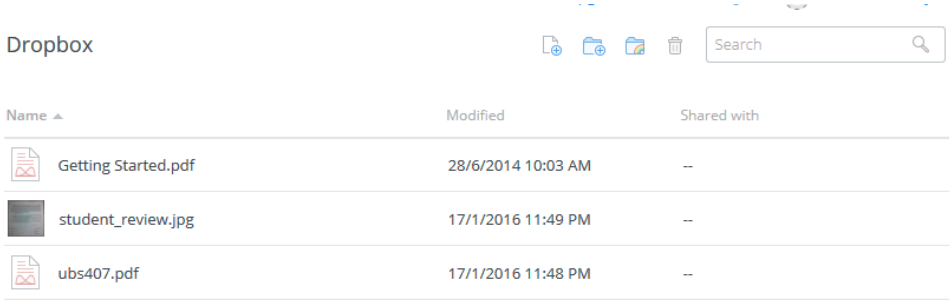

Рисунок 4.2 – Файли, розташовані в сервісі Dropbox

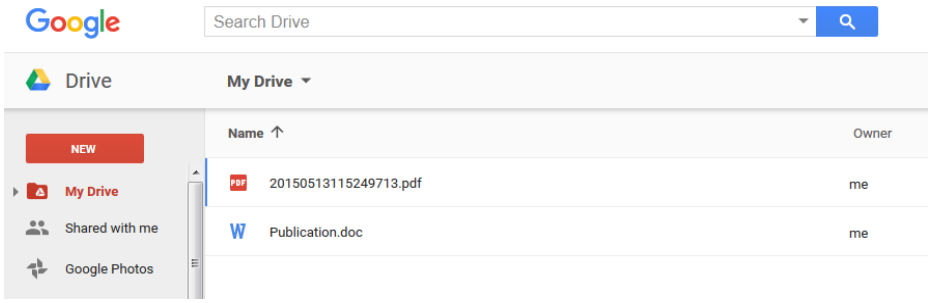

Рисунок 4.3 – Файли, розташовані в сервісі Google Drive

Так само, на головнім вікні відображається кількість загального місця й використаного.

При натисканні на кнопку «Browse» відкривається вікно з вибором файлу. При натисканні на кнопку «Upload» файл завантажується на обраний хмарний диск. При натисканні на ім'я файлу відкривається вікно завантаження файлу. При натисканні на кнопку «Delete» відповідний файл віддаляється. При натисканні на кнопку «Move to…» файл переноситься на соответсвующее сховище. На рисунках 4.4, 4.5, 4.6 показані головне вікно програми, Dropbox і Google Drive відповідно після завантаження нового файлу, видалення старого й переміщення.

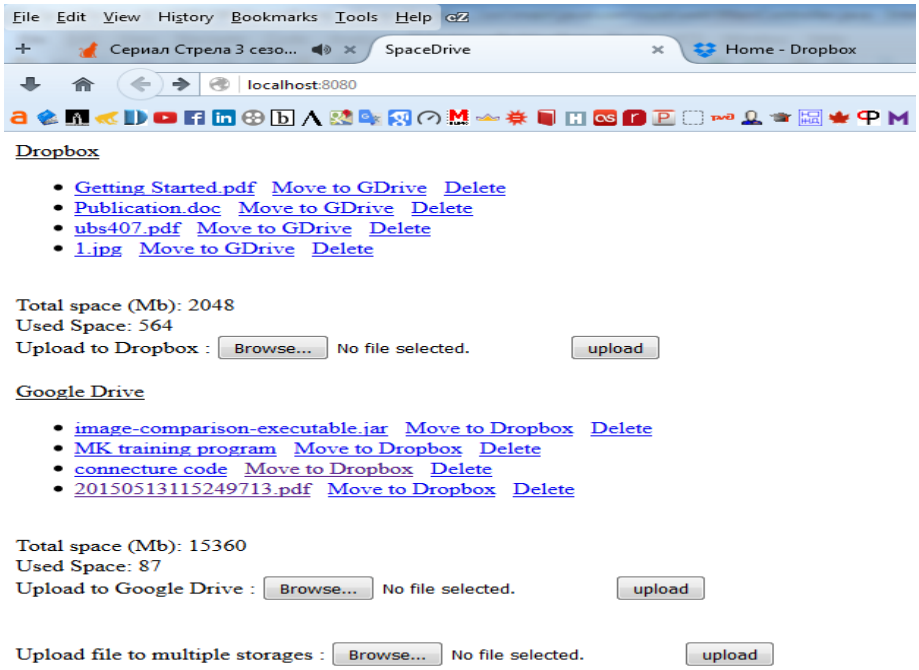

Рисунок 4.4 – Головне вікно програми після зміни файлів

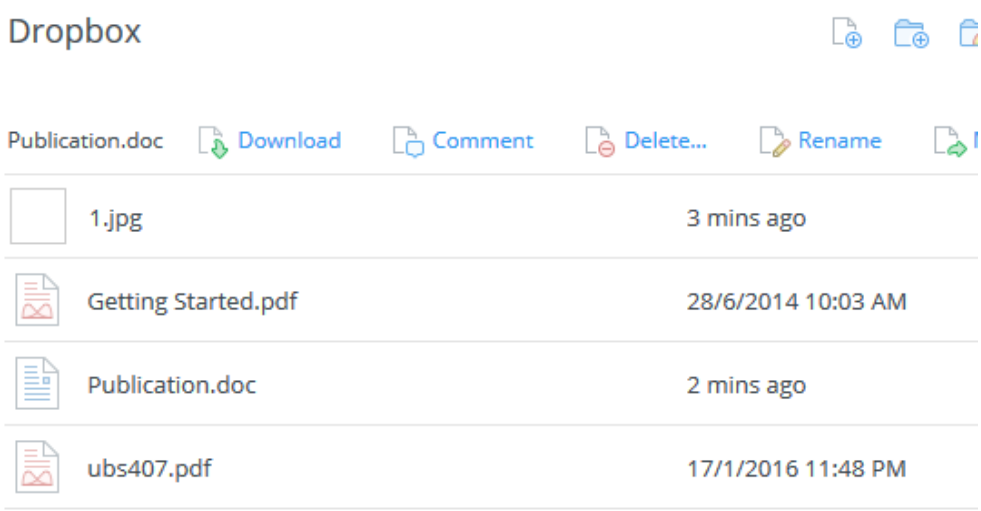

Рисунок 4.5 – Сервіс Dropbox після зміни файлів

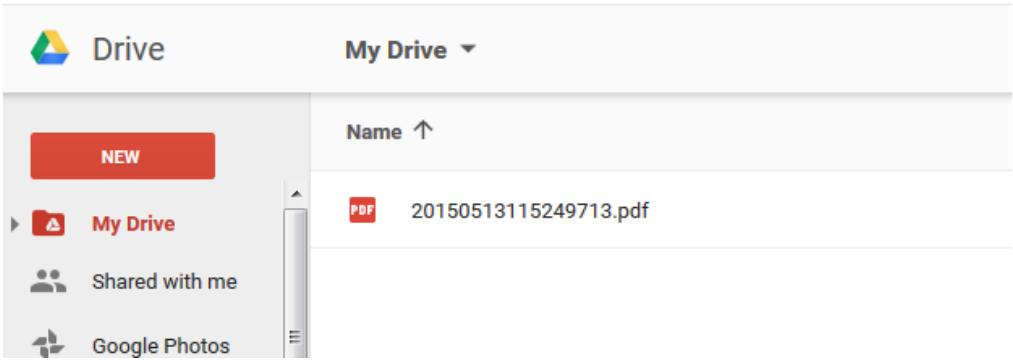

Рисунок 4.6 – Сервіс Google Drive після зміни файлів

Якість роботи програми було перевірено на прикладі інтеграції хмарних сховищ Dropbox і Google Drive. Обоє сховища були обрані випадковим образом.

Був проведений ряд експериментів, у яких перевірялися можливість і зручність інтеграції й централізованого використання різних хмарних сховищ.

5 ОПИС МОЖЛИВОСТІ ВИКОРИСТАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

У ході експерименту було проведені дії по скачиванию файлів із хмарних сховищ, їх завантаженню, видалення, а так само переміщення з одного сховища на інше й можливості розбивки файлу на кілька частин, завантаження цих частин у сховища й наступне скачивание цілісного файлу. Усі дані дії були зроблені успішно, що підтверджує можливість інтеграції хмарних сховищ.

Таким чином, була описана й спроектована комп'ютерна програма для інтеграції хмарних сховищ. Програма була написана мовою високого рівня Java. Програма дозволяє виконувати базові дії, доступні будь-якому користувачеві хмарного сховища, такі як скачивание, видалення, завантаження, а так само дозволяє виконувати дії, які доступні тільки при використанні інтеграційної платформи, такі як переміщення файлів між сховищами, розбивка файлу на частині й завантаження на різні сховища.

Програма може працювати не тільки зі сховищами, які використовувалися в ході експерименту.

Використання отриманих результатів можливо як у науковій, так і в практичній діяльності. Незважаючи на те, що експеримент дав позитивні результати, є можливість дослідження можливості паралельного завантаження файлів на кілька хмарних сховищ, а так само можливість дефрагментации файлів у сховищах. У практичній діяльності, отримані результати можуть використовуватися як інтеграційна платформа для особистих потреб компаній, так і як незалежна інтеграційна платформа, доступна для використання звичайними користувачами.

Для нормальної експлуатації системи необхідно мати ЕОМ і інші технічні засоби з наступними характеристиками:

- процесор: AMD Atlon 1000 або вище;
- оперативна пам'ять - 1Gb або більш;

– вільний дисковий простір: 200 Mb для програмних модулів, що входять даних і збереження результатів виконання програми;

– операційна система: Windows 10, *Unix;

– клавіатура й комп'ютерна миша;

– відеопідсистема з можливістю відображення графічної інформації в дозволи 1024x768 пикселей або вище;

– підключення до мережі інтернет.

ВИСНОВКИ

Дана робота присвячена темі розробки в області веб-технологи «технологічний стік» у застосуванні до веб-програмуванню й розробці на її основі алгоритму інтеграції хмарних сховищ.

У роботі були розглянуті поняття «технологічний стік», «хмарного сховища», інтеграційної платформи, хмарного брокера. Був проведений аналіз і стан проблеми розробки веб-додатків за допомогою технологічного стека, інтеграції хмарних сховищ, аргументована актуальність проведеного дослідження. Розглянуті проблеми інтеграції й безпеки хмарних сховищ, а так само їхньої переваги й недоліки. Проаналізовані способи захисту, архітектурні рішення й оптимізації обробки даних, вироблених при проектуванні хмарних сховищ.

Розглянуті платформи інтеграції хмарних сховищ і запропонована нова модель, яка надає більший функціонал і зручності, чому наявні на ринку аналоги, з використання більш швидкого алгоритму інтеграції даних.

Розроблене програмне забезпечення для інтеграції й синхронізації даних на різних хмарних сховищах. Програма Spacedrive призначена для роботи на одній ЕОМ і реалізована мовою високого рівня Java. Запропонована архітектура обладнання для використання різних хмарних сховищ як одне ціле.

Поведене дослідження і його результати можуть бути використані в комерційних цілях для зменшення витрат бізнесу на впровадження власних рішень для інтеграції хмарних сервісів, підтримку цих рішень, навчання персоналу роботи з різними хмарними сервісами.

Результати роботи:

- алгоритм інтеграції даних у хмарних сховищах;
- технологічний стік для хмарної інтеграції даних;
- комп'ютерна програма, випробувана на реальних прикладах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Визинг В. Г. Хроматический класс мультиграфа // Кибернетика. – 1965. - №3. - С. 29-39

2. Евстигнеев В. П. Графи в программировании: обработка, визуализация и применение / В. П. Евстигнеев, В. Г. Касьянов; - С.-Пб. : БХВ-Петербург, 2013. - 1104 с.

3. Риз Дж. Облачние вичисления / Дж. Риз. - С.-Пб. : БХВ-Петербург, 2011. - 288 с.

4. Змитрович А. И. Интеллектуальние информационние системи / А. И. Змитрович. - М. : ТетраСистемс, 2007. - 349 с.

5. Олейников А.Я. Технология откритих систем [/ А. Я. Олейников. - М. : Янус-К, 2004. - 286 с.

6. Макаров С.В. Социально-экономические аспекти облачних вичислений / С. В. Макаров. - М. : Реал-бук, 2010. - 120 с.

7. Липский, В. А. Комбинаторика для программистов / В. А. Липский. – М. : Мир, 1988. - 213 с.

8. Денисов В. Н. Перспективи развития облачних вичислений / В. Н. Денисов – М. : Синергия, 2007. – 144 с.

9. Мельников В. П. Информационная безопасность / В. П. Мельников, - М. : Academia, 2013. – 317 с.

10. Колегов Д. С. Анализ безопасности управления доступом в компьютерних системах / Д. С. Колегов, -М. : LAP Lambert Academic Publishing, 2011. – 485 с.

11. Игошин В. К. Теория алгоритмов/ В. К. Игошин, -М. : Academia, 2013. – 332 с.

12. Макконнелл Д. Анализ алгоритмов. Активний обучающий подход / Дже. Макконнелл, -М. : Техносфера, 2009. – 587 с.

13. Арсеньев Б. А. Интеграция распределенних баз данних/ Б. А. Арсеньев,

С. Ф. Яковлев, -К. : Лань, 2018. – 350 с.

14. Оптимизация облачного хранения данних / URL: http://www.ibm.com/developerworks/ru/library/cl-optimize-data-storage-cloud/ - -.

15. Анатомия облачной инфраструктури хранения данних URL: http://www.ibm.com/developerworks/ru/library/cl-cloudstorage/.

16. Волков, А. Д. Интеграция хранилищ данних с откритими и большими данними для решения задач финансовой организации: проблеми и подходи к решению / А. Д. Волков, -М.: Синергия, 2008, - 428 с.

17. Виноградов, В. В. Информационно-вичислительние системи. Распределенние модульние системи автоматизации / В. В. Виноградов, -К : БХВ, 2016, - 336 с.

18. Гибридная интеграция / Откритие системи. СУБД. - URL: http://www.osp.ru/os/2019/12/13043484/.

19. Оптимальний алгоритм миграции данних в масштабируемих облачних хранилищах / URL: http://cyberleninka.ru/article/n/optimalnyy-algoritm-migratsii-dannyh-v-masshtabiruemyh-oblachnyh-hranilischah.

20. Методи и платформи интеграции данних и организации хранилищ больших данних URL: http://synthesis.ipi.ac.ru/synthesis/student/BigData/seminar-integration

21. Облачное хранилище данних / URL: http://www.leonsky.net/smart-storage.

22. Уоррен Г. Алгоритмические трюки для программистов / Генри Уоррен, -М : Вильямс, 2014. – 479 с.

23. Седжвик Р. Алгоритми на Java / Р. Седжвик, -М : Вильямс, 2015. – 848 с.

24. Шилдт Г. Java 8. Полное руководство / Г. Шилдт, -М : Вильямс, 2015. – 1376 с.

25. Система и способ записи данних в облачное хранилище URL: http://www.findpatent.ru/patent/243/2435236.html.

26. Уоллс, К. Spring в действии / К. Уоллс, -М : ДМК Пресс, 2015 – 932 с.

27. Зиков А. А. Основи теории графов / А. Зиков, -К : БХВ. 2018 - 526 с.