МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Завідувач кафедри
_____ Скарга-Бандурова І.С.
«____»_____ 2019 р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

Дослідження та розробка програмно-технічних засобів розподіленої системи обробки відеоданих

Освітньо-кваліфікаційний рівень "Магістр"
Спеціальність 123 "Комп'ютерна інженерія"

Науковий керівник роботи:	_____ (підпис)	О.І.Рязанцев (ініціали, прізвище)
Консультант з охорони праці:	_____ (підпис)	Я.О.Критська (ініціали, прізвище)
Студент:	_____ (підпис)	Ю.А. Мельник (ініціали, прізвище)
Група:		КІ-17дм

Сєвєродонецьк 2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерноих наук та інженерії
Освітньо-кваліфікаційний рівень "магістр"
Спеціальність 123 – "Комп'ютерна інженерія"
(шифр і назва)
Спеціалізація
(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри КНІ
д.т.н., доц. *І.С. Скарга-Бандурова*
«_____» _____20____р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Мельнику Юрію Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розробка програмно-технічних засобів розподіленої системи обробки відеоданих

керівник проекту (роботи) Рязанцев О.І., д.т.н., професор
(прізвище, ім.'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «18› 10 2018 р. № 220/48
2. Строк подання студентом роботи
3. Вихідні дані до роботи Матеріали науково-дослідної практики.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд сучасного стану систем обробки даних.
2. Дослідження методу розподіленої обробки даних.
3. Методи сегментації відео.
4. Метод кодування відео.
5 Результати моделювання та експериментів.
6.Охорона праці та безпека в надзвичайних ситуаціях.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Електронні плакати

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Критська Я.О.*		

7. Дата видачі завдання *20.10.2018*

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Формування технічного завдання	*21.10.18-25.10.18*	
2	Аналіз завдання, робота з літературою	*26.10.18-03.11.18*	
3	Дослідження в області розробки системи	*04.11.18-14.11.18*	
4	Розробка програмної системи	*15.11.18-30.11.18*	
5	Аналіз результатів аналізу	*01.12.18-14.12.18*	
6	Тестування програмної системи	*15.12.18-19.12.18*	
7	Охорона праці	*20.12.18-21.12.18*	
8	Оформлення пояснювальної записки	*22.12.18-28.12.18*	
9	Оформлення комп'ютерної презентації	*02.01.19-18.01.19*	

Студент _____ *Мельник Ю.А.*
(підпис) (прізвище та ініціали)

Науковий керівник _____ *Рязанцев О.І.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мельник Ю.А. Дослідження та розробка програмно-технічних засобів розподіленої системи обробки відеоданих.

Метою магістерської роботи є дослідження методів обробки та сегментації відео та розробка системи для збереження та кодування мультимедійного проекту, керування їм; та системи керування медіа файлами проекту. Результатом роботи є система, яка вирішує проблему швидкісної обробки відео.

Ключові слова: HEVC, сегментація, кодування, розподілена обробка, програмна модель, тестування.

АННОТАЦИЯ

Мельник Ю.А. Исследование и разработка программно-технических средств распределенной системы обработки видеоданных.

Целью магистерской работы является исследование методов обработки и сегментации видео и разработка системы для сохранения и кодирования мультимедийного проекта, управления им; и системы управления медиа файлами проекта. Результатом работы является система, которая решает проблему скоростной обработки видео.

Ключевые слова: HEVC, сегментация, кодирования, распределенная обработка, программная модель, тестирование.

ABSTRACT

Melnik Yu.A. Research and development of software and hardware for a distributed video data processing system.

The aim of certification diploma is to study the video processing and segmentation methods and the development of a system for storing and coding a multimedia project, managing it; and project media management systems. The result of the work is a system that solves the problem of high-speed video processing.

Key words: HEVC, segmentation, coding, distributed processing, program model, testing.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СКВ – Система керування версіями

ЦСКВ – Централізовані системи контролю версій

ОС – Операційна система

LBT – Локальний багтрекер

СОВ – Система обробки версій

ВСТУП

Зараз у мультимедійній індустрії є недолік з точки зору, що немає централізованої системи керування проектами, під чим мається на увазі, що сучасні мультимедійні проекти неможливо зручно зберігати на серверах, переглядати історію змін, тощо. Також є проблема обчислення відео. Відеодані стає складніше і складніше обробляти тому, що формати відео розвиваються швидше ніж зростає потужність машин.

Для вирішення цієї проблеми пропонується створити систему, яка буде зберігати мультимедійний проект, керувати їм та обробляти його розподіллено. Робити цей процес, використовуючи локальне обладнання, не є правильним. Так як реальне використовування цього обладнання буде лиш у 5-10% від усього часу. Набагато розумнішим буде використовувати для цього кластер з серверів, що дасть змогу використовувати наше обладнання 100% часу та значно швидше.

Це дає значну економію ресурсів, і завдяки цьому можно направити їх в інші напрями виробництва.

Також слід розробити зручну систему керування проектом, а більш точніше – системи керування медіа файлами проекту. Для цього пропонується розробити систему керування версіями для медіа файлів, а потім використовувати для їх збереження на сервері, та взаємодії між людьми.

1 СИСТЕМИ РОЗПОДІЛЕННОГО ЗБЕРІГАННЯ ДАНИХ

1.1 Поняття системи керування версіями

Система керування версіями – це система, що записує зміни у файл або набір файлів протягом деякого часу, так що можливо повернутися до певної версії пізніше [1]. Як приклад, для файлів, що знаходяться під контролем версій, буде використовуватися код програмного забезпечення, хоча насправді можно використовувати контроль версій практично для будь-яких типів файлів.

Система контролю версій дозволяє повернути файли до стану, в якому вони були до змін, повернути весь проект до попереднього стану, побачити зміни, побачити, хто останній міняв щось і спровокував проблему, хто вказав на проблему і коли та багато іншого. Використання СКВ також в цілому означає, що, якщо зламали щось або втратили файли, просто можливо все виправити.

1.2 Локальні системи контролю версій

Багато людей в якості одного з методів контролю версій застосовують копіювання файлів в окрему директорію (можливо навіть директорію з відміткою за часом, якщо вони достатньо розумні). Даний підхід є дуже поширеним завдяки його простоті. Проте він неймовірним чином схильний до появи помилок. Можна легко забути, в якій директорії знаходитеся і випадково змінити не той файл або скопіювати не ті файли, які хотіли.

Щоб справитися з цією проблемою, програмісти давно розробили локальні СКВ, що мають просту базу даних, яка зберігає всі зміни в файлах під контролем версій [2]. На рисунку 1.1 зображено відображення локальної системи контролю версіями.

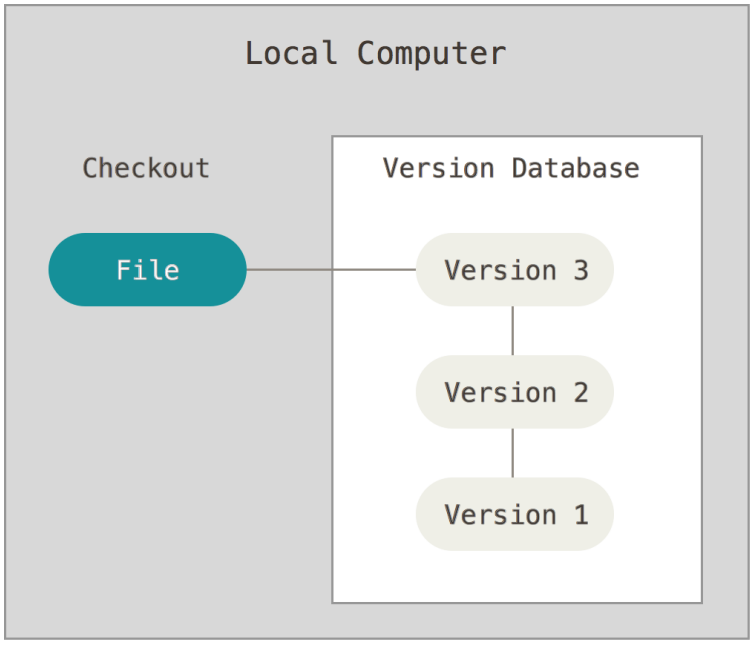

Рисунок 1.1 – Локальні системи контролю версій

Одним з найбільш поширених інструментів СКВ була система під назвою RCS, яка досі поширюється з багатьма комп'ютерами сьогодні. Навіть популярна операційна система Mac OS X включає в себе команду rcs при інсталяції Developer Tools.

RCS – це одна з перших реалізацій системи керування версіями. Для кожного файлу, зареєстрованого в системі, вона зберігає повну історію змін. Для ефективного використання дискового простору при зберіганні текстових файлів використовується алгоритм дельта-компресії, коли зберігається тільки перша версія і всі міжверсійні зміни. Система дозволяє також зберігати версії бінарних файлів, але без використання цього механізму, тобто кожна версія бінарного файлу зберігається повністю.

Система RCS не має засобів для колективної роботи над набором файлів – ці засоби з'явилися в системі-спадкоємниці – CVS, що використовує формати і алгоритми RCS для обліку версій, але має також інтерфейси для колективної роботи.

Відсутність колективної роботи на практиці виглядає так, що зміни може вносити тільки той користувач, який заблокував файл, всі інші мають очікувати доки зміни не будуть збережені або файл не буде розблоковано [3].

Дельта-кодування – спосіб представлення даних у вигляді різниці (дельти) між послідовними даними замість самих даних. Мабуть, найбільш простий приклад полягає в збереженні значень байтів як відмінності (дельти) між послідовними значеннями, на відміну від самих значень. Тому замість 2, 4, 6, 9, 7, ми будемо зберігати 2, 2, 2, 3, -2. Це не дуже корисно в разі, коли використовується саме по собі, але може допомогти в разі

подальшої компресії цих даних, в яких часто зустрічаються повторювані значення. Алгоритми компресії часто вибирають дельта-кодування тільки тоді, коли стиснення з ним краще, ніж без нього. Також слід зазначити, що в стисненні відео дельта-фрейми можуть значно зменшувати розмір фрейму, тому вони використовуються практично в кожному відеокодеку.

У дельта-кодованої передачі по мережі, де тільки одинична копія файлу доступна на кожному кінці комунікаційного каналу, використовуються спеціальні коди корекції помилок для виявлення того, які частини файлу змінилися з часу попередньої версії.

Дельта-кодування застосовується як попередній етап для багатьох алгоритмів стиснення, наприклад RLE, і в інвертованих індексах пошукових програм. Природа даних, які будуть закодовані, значно впливає на ефективність стиснення. Дельта-кодування підвищує коефіцієнт стиснення в тому випадку, коли дані мають маленьку або постійну варіацію (як, наприклад, градієнт на зображенні); для даних, згенерованих генератором випадкових чисел з рівномірним розподілом, коефіцієнт стиснення зміниться незначно.

Дельта-кодування робить неможливим довільний доступ до даних, так як для звернення до елементу масиву необхідно підсумувати значення всіх попередніх. Якщо це все ж необхідно, застосовується блоковий варіант дельта-кодування, в якому кодуються блоки деякої заданої довжини. Тоді необхідно лише підсумувати значення з початку блоку, до якого належить шуканий елемент, але не всього файлу. Розмір блоку вибирається в залежності від програми, зазвичай за результатами хронометражу [4].

1.3 Централізовані системи контролю версій

Наступним важливим питанням, з яким стикаються, є необхідність співпрацювати з іншими розробниками. Щоб справитися з цією проблемою, були розроблені централізовані системи контролю версій. Такі системи як CVS, Subversion і Perforce мають єдиний сервер, який містить всі версії файлів, та деяке число клієнтів, які отримують файли з центрального місця. Протягом багатьох років, це було стандартом для систем контролю версій [2]. На рисунку 1.2 зображено схему централізованої системи контролю версіями.

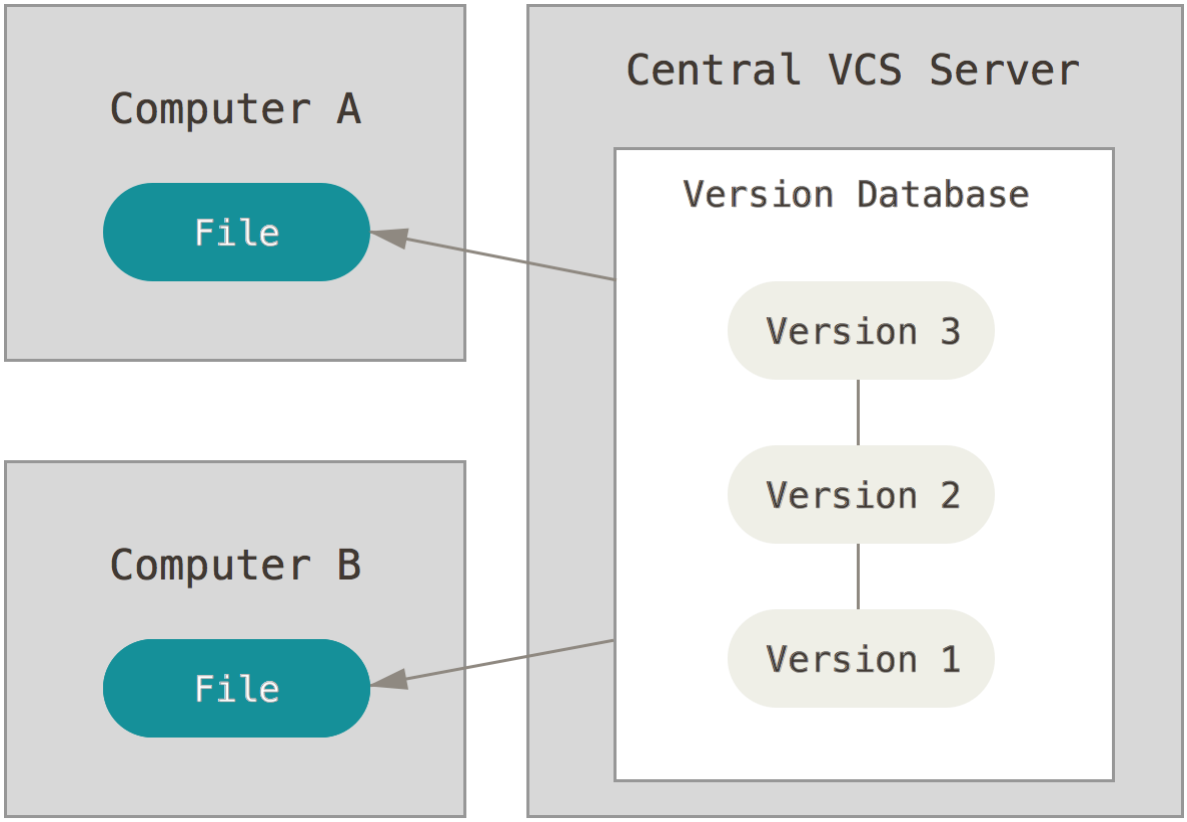

Рисунок 1.2 – Централізовані системи контролю версій

Такий підхід має безліч переваг, особливо над локальними СКВ. Наприклад, кожному учаснику проекту відомо, певною мірою, чим займаються інші. Адміністратори мають повний контроль над тим, хто і що може робити; і це набагато легше адмініструвати з ЦСКВ, ніж мати справу з локальними базами даних для кожного клієнта.

Але цей підхід також має деякі серйозні недоліки. Найбільш очевидним є єдина точка відмови, яким є централізований сервер. Якщо сервер виходить з ладу протягом години, то протягом цієї години ніхто не може співпрацювати або зберігати зміни, над якими вони працюють під версійним контролем. Якщо жорсткий диск центральної бази даних на сервері пошкоджено, і своєчасні резервні копії не були зроблені, втрачається абсолютно все – вся історія проекту, крім одиночних знімків проекту, що збереглися на локальних машинах. Локальні СКВ страждають тією ж проблемою – щоразу, коли вся історія проекту зберігається в одному місці, є ризик втратити все.

1.3.1 Централізована система керування версіями CVS

CVS використовує архітектуру клієнт-сервер: сервер зберігає поточну версію (версії) проекту і його історії, і клієнти підключаються до сервера, щоб загрузити повну копію проекту, виповнити роботу над цією копією, а потім пізніше загрузити на сервер їх зміни. Як правило, клієнт і сервер підключенні через локальну мережу або через Інтернет, але клієнт і сервер можуть працювати як на тій же машині, якщо CVS має завдання відстежування історії версій проекту тільки з місцевими розробниками. Програмне забезпечення сервера зазвичай працює на Unix (хоча, сервер CVSNT також підтримує різні версії Microsoft Windows), в той час як клієнти CVS можуть працювати на будь-якій операційній системі [5].

Кілька розробників можуть працювати над одним проектом одночасно, кожен з яких редагує файли в межах своєї власної "робочої копії" проекту, і відправляти (або перевіряти) їх на сервер. Для того, щоб уникнути можливості взаємозамінних змін, сервер приймає тільки зміни, зроблені в самій останній версії файлу. Тому очікується, що розробники будуть завжди виконувати роботу з останньою копією проекту. Це завдання в основному виконується автоматично клієнтом CVS, вимагаючи ручного втручання тільки тоді, коли виникає конфлікт між локальними файлами та файлами на сервері.

Якщо перевірка завершується успішно, то номери версій всіх файлів, які беруть участь в цьому, автоматично збільшуються і CVS-сервер записує наданий користувачем рядок опису, дату і ім'я автора в його лог-файли. CVS також може запускати зовнішні, задані користувачем скрипти обробки логів після кожного комміта. Ці сценарії встановлюються за допомогою запису в файлі LOGINFO CVS, який може викликати повідомлення по електронній пошті або перетворити дані журналу до Web-based формату.

Клієнти також можуть порівняти версії, запросити повну історію змін, або перевірити історичний знімок проекту станом на певну дату або за станом на номер ревізії.

Лог – спеціальний файл, в якому накопичується зібрана службова та статистична інформація про події в системі (програмі). Операційні системи (особливо це стосується серверних ОС) та серверне програмне забезпечення зазвичай мають розвинуту систему ведення логів. За допомогою них різні типи подій, різну інформацію можна зберігати у своїх спеціалізованих логах.

Зокрема логи сайтів зазвичай накопичують і зберігають службову інформацію про відвідувачів (IP-адресу комп'ютера відвідувача, коли відвідувався сайт і скільки часу користувач провів на сайті, яку інформацію переглядав та копіював, який браузер використовувався тощо).

Інформація з лог-файлів надалі використовується адміністраторами для аналізу подій, виявлення помилок, збоїв, зведення статистики, звітування, стеження дій підозрілих користувачів, вузлів тощо.

Комміт – це операція, яка використовується для збереження стану проекту у централізованих та децентралізованих системах контролю версіями. Це означає, що у будь-який час є спроможність повернутися до будь-якого комміту або порівняти комміт з локальною версією.

Комміт містить унікальний хеш для ідентифікації, дельту кожного зміненого файлу та мета данні для ідентифікації автора [2].

1.3.2 Централізована система керування версіями Subversion

Subversion розроблена спеціально для заміни CVS, найпоширенішої відкритої системи управління версіями. Вона має всі основні функції CVS (хоча деякі з них виконує іншими способами) і вільна від ряду її недоліків. Subversion часто називають «svn», по назві клієнтської програми, що входить в її дистрибутив[6].

Слід зазначити наступні відмінності від CVS:

– Subversion відстежує версії не тільки файлів, але і каталогів;

– якщо зміни зроблені в декількох файлах і каталогах, вони публікуються як одна транзакція. Це означає, що або в сховищі потрапляють всі зміни, або стан сховища не змінюється;

– при будь-яких оновленнях версій між клієнтом і сервером передаються тільки відмінності між файлами;

– Subversion підтримує копіювання, переміщення і перейменування файлів із збереженням історії змін;

– з кожним файлом і каталогом може бути зв'язаний довільний набір властивостей, що складаються з назви і значення. Властивості теж знаходяться під управлінням версіями;

– Subversion однаково ефективно працює як з текстовими, так і з двійковими файлами;

– починаючи з версії 1.2, підтримується необов'язкове блокування файлів;

– Subversion немає міток (tag) і гілок (branch), як таких. Замість них використовується ієрархія каталогів – для кожної гілки або мітки створюється окремий каталог. Створення таких каталогів – швидка і дешева операція, тому що дані не дублюються, натомість публікується нова версія, що відрізняється від попередньої лише розташуванням файлів.

Мітка, або англійською tag, зазвичай застосовується для позначення версій продукту. Мітка вказує нам на специфічний комміт та значно спрощує роботу з СКВ. Так як зазвичай кожна версія містить у собі поряд 10-20 коммітів у проектах з одним розробником. 50-100 на проектах, на яких до 5 розробників. А на великих проектах кожна версія може містити у собі декілька сотень коммітів. Наприклад, можемо у будь-який час порівняти між собою дві версії продукту.

Гілка, або англійською branch, використовується для простішої взаємодії на проектах та задля забезпечення стабільності. Гілка представляє собою відгалуження від якогось комміта, та розробка ведеться паралельно з іншими гілками без блокування процесу. Це потрібно у разі, якщо проміжні зміни є нестабільними, але бажано зберегти і їх. Наприклад, є головна гілка, вона є завжди, та перший комміт, ініціалізацію будемо проводити у неї. Потім бажаємо додати деякі нові функції до цієї системи. Для цього створюємо гілку, яка буде використовуватися тільки для однієї функції, та можемо експериментувати там так багато, як бажаємо, і можемо зберігати будь-який стан розробки не турбуючись за те, що інший розробник буде використовувати це, як основу для своєї роботи. Коли робота буде закінчена, ці зміни будуть додані до основної гілки, і всі зможуть скористатися останньою стабільною версією проекту, при цьому з доступом до усіх проміжних змін.

Subversion є самою розповсюдженою централізованою системою керування версіями.

1.3.3 Централізована система керування версіями Perforce

Це комерційна СКВ, яка має властивості CVS та Subversion, а також має у собі влаштований багтрекер. Серед недоліків слід зазначити те, що на відміну від попередніх двох, які розповсюджується під open-source ліцензіямі, Perforce розповсюджується по комерційній ліцензії, що і стало причиною такої низької популярності [7].

Баг – це збій або помилка в комп'ютерній програмі або системі, яка змушує його давати неправильний або несподіваний результат, або вести себе непередбачуваним чином. Більшість багів виникають з помилок, зроблених людьми в будь-якому вихідному коді програми або його конструкції, або в рамках операційних систем, які використовуються такими програмами, а деякі з них викликані компіляторами, які виробляють неправильний код. Програма, яка містить велику кількість помилок, і / або помилки, які серйозно заважають функціональності, називається блокуючою. Детальні звіти про помилки в програмі, як правило відомі як повідомлення про помилки [8].

Багтрекер – прикладна програма, розроблена, щоб допомогти тестерам та програмістам відстежувати історію звітів про баги під час своєї роботи.

Багато багтрекерів, зокрема ті, що використовуються більшістю open source проектів, дозволяють користувачам вводити звіт про помилку безпосередньо. Інші системи використовуються лише всередині компаній чи організацій, що займаються розробкою програмного забезпечення [8].

Наявність багтрекера вкрай важлива у розробці програмного забезпечення. Вони широко використовуються компаніями, що розробляють програмні продукти. Послідовне використання багтрекера вважається однією з «ознак хорошої команди програмістів».

Головний компонент багтрекера – база даних, що записує факти про відомі баги. Факти можуть включати час звіту про баг, його серйозність, неправильну поведінку програми, деталі про те, як відтворити помилку, а також особу, що повідомила про помилку, та програмістів, котрі могли працювати над її виправленням.

Типові багтрекери підтримують концепцію життєвого циклу бага, що відстежується через статус, який присвоєний багу. Багтрекер дозволяє адміністраторам конфігурувати права на основі статусу, змінювати статус бага чи вилучати баг. Система також дозволяє адміністратору конфігурувати статуси багів і до якого статусу баг може бути змінено в кожному окремому випадку. Деякі системи надсилають електронного листа зацікавленим сторонам, таким як submitter та assigned програмісти, коли додається новий запис чи змінюється статус.

Головна перевага багтрекера полягає в забезпеченні чіткого централізованого огляду запитів розробки (включаючи як баги, так і зручності, різниця часто нечітка), і їх стану. Список пріоритетів незавершених пунктів (що часто називається backlog) забезпечує вагомий вклад при визначенні перспективного плану продукту чи просто «наступного релізу».

У корпоративному середовищі багтрекер може використовуватись, щоб генерувати звіт з продуктивності програмістів у виправленні багів. Однак, це може інколи спричинити неточний результат через те, що різні баги можуть мати різні рівні серйозності й складності. Серйозність бага не може безпосередньо пов'язуватись зі складністю виправлення бага. Погляди менеджерів та архітекторів можуть відрізнятись.

Локальний багтрекер – звичайна комп'ютерна програма, використовувана командою професійних підтримувачів додатку (часто служба технічної підтримки), відстежування помилок, зв’язаних з розробниками програмного забезпечення. Використання LBT дозволяє спеціалістам зі служби підтримки відстежувати баги на «своїй власній мові», а не на «мові розробників». Крім того, використання LBT дозволяє команді підтримки відстежувати інформацію про користувачів, що висловили скарги, неважливі для процесу розробки (таким чином, коли використовується LBT, існує два багтрекери) [8].

Деякі багтрекери розроблено для роботи з програмами розподіленого контролю версій. Ці розподілені багтрекери дозволяють легко читати, додавати до бази даних чи оновлювати звіти про помилки, поки розробник не в мережі.

Відкрите програмне забезпечення (англ. open-source software) – програмне забезпечення з відкритим сирцевим кодом.

Сирцевий код таких програм доступний [9]:

– для перегляду і вивчення;

– за наявності дозволу ліцензії – зміни, що дозволяє користувачеві узяти участь у доопрацюванні відкритої програми;

– використовувати код для створення нових програм – через запозичення сирцевого коду, якщо це дозволяє сумісність ліцензій;

– виправляти в ній помилки;

– вивчення використаних алгоритмів, структур даних, технологій, методик та інтерфейсів (оскільки сирцевий код може істотно доповнювати документацію, а за відсутності такої сам служить документацією).

1.4 Децентралізовані системи контролю версій

До децентралізованих (розподілених) систем контролю версій відносятся Git, Mercurial, Bazaar або Darcs. В цьому випадку клієнти не просто отримують останній знімок файлів репозиторію: вони повністю відображають сховище. Таким чином, якщо вмирає який-небудь сервер, через який співпрацюють розробники, будь-який з клієнтських репозиторіїв може бути скопійований назад до серверу, щоб відновити його. Кожна копія дійсно є повною резервної копією всіх даних [2]. На рисунку 1.3 зображено схему децентралізованої системи контролю версіями.

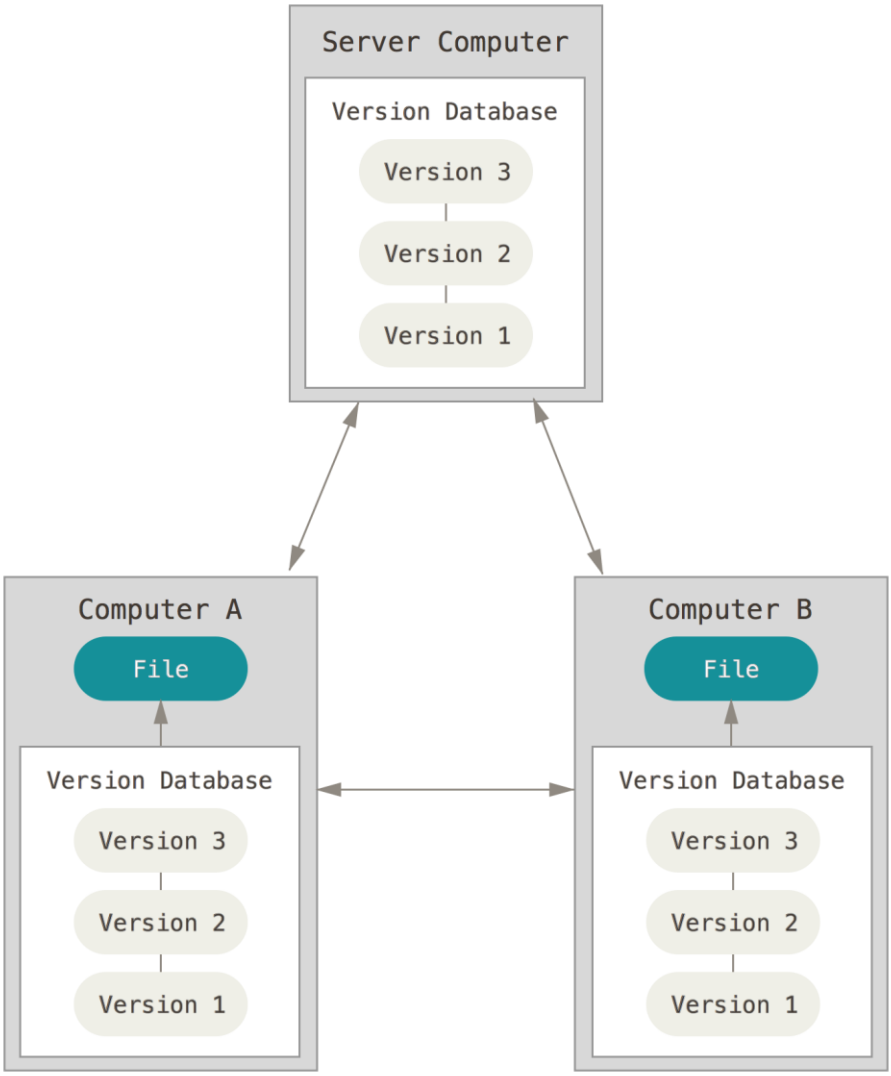

Рисунок 1.3 – Децентралізовані системи контролю версій

Більш того, багато з цих систем дуже добре взаємодіють з декількома віддаленими репозиторіями так, що можливо співпрацювати з різними групами людей, застосовуючи різні підходи в межах одного проекту одночасно. Це дозволяє налаштувати декілька типів робочих процесів, таких як ієрархічні моделі, які неможливі в централізованих системах. Децентралізовані системи дуже популярні у наш час ще з тих причин, що ми бажаємо звести час, в який не маємо можливості працювати до мінімуму. Тому дуже зручно використовувати децентралізовані системи, так як вони дозволяють працювати навіть без доступу до інтернету, так як у завжди є локальна копія усього [2].

1.4.1 Розподілена система керування версіями Mercurial

Mercurial – вільна розподілена система керування версіями файлів та спільної роботи, розроблена для ефективної роботи з дуже великими репозиторіями сирцевого коду. Mercurial спочатку був написаний для Linux та пізніше портований під Windows, Mac OS X і більшість Unix-систем. У першу чергу він є консольною програмою. Всі його операції запускаються параметрами програми hg, назва якої походить від позначення хімічного знака ртуті (англ. mercury)[10].

Із переваг Mercurial можна відмітити:

– швидкодія;

– висока продуктивність роботи з сховищем, незалежна від числа елементів у ньому;

– компактне зберігання даних в проіндексованому і стислому вигляді;

– оптимізований для ефективної роботи з даними на твердому диску;

– всі зміни та файли в сховищі додатково проіндексовані;

– для копіювання даних по мережі використовується HTTP і SSH, дані передаються в стислому вигляді.

Масштабування:

– розподілена модель розробки дозволяє брати участь у проекті необмеженому числу розробників;

– допускається довільне злиття окремих децентралізованих сховищ, які підтримуються окремими розробниками;

– обсяг сховища, число файлів і зафіксованих змін не відбивається негативно на продуктивності;

– при роботі немає потреби очікувати звільнення блокування.

Надійність:

– для контролю цілісності даних в сховищі використовується алгоритм SHA1;

– сховище реалізовано в журнальному вигляді – дані не заміщаються, а додаються. Ведеться журнал транзакцій;

– швидкий алгоритм перевірки цілісності сховища;

– вбудовані засоби резервного копіювання та перевірки цілісності.

Зручність використання;

– звичний CVS-подібний набір команд;

– наявність вбудованої системи підказки;

– інтегрований веб-сервер, який дозволяє мати доступ до сховища через веб-інтерфейс.

Легкість впровадження:

– підтримка платформ UNIX і Windows;

– засоби, що спрощують міграцію з інших систем управління вихідними текстами;

– підтримка декількох моделей організації сховища;

– підтримка зовнішніх обробників і доповнень.

1.4.2 Розподілена система керування версіями Git

Git – розподілена система керування версіями файлів та спільної роботи. Проект створив Лінус Торвальдс для управління розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів [2].

Ядро Linux – це проект досить великого обсягу з відкритим програмним кодом. Більшу частину часу підтримання ядра Linux (1991-2002) виконувалося у вигляді патчів та

архівів. У 2002 році проект ядра Linux почав використовувати закриту систему контролю версій BitKeeper.

У 2005 році відносини між спільнотою розробників ядра Linux і комерційною компанією, що розробила BitKeeper почали псуватись, і безкоштовне використання продуктом було скасовано. Це підштовхнуло розробників Linux (і зокрема Лінуса Торвальдса, автора Linux) розробити власну систему, ґрунтуючись на деяких з уроків, які вони дізналися під час використання BitKeeper. Деякі з цілей нової системи були:

– швидкість;
– проста архітектура;
– сильна підтримка для нелінійного розвитку (тисячі паралельних гілок);
– децентралізація;
– можливість ефективно управляти великими проектами, такими як ядро Linux (швидкість і розмір даних).

1.5 Архітектура Git

1.5.1 Основи Git

Головна відмінність від інших систем (таких як Subversion та подібних їй) є те, як Git сприймає дані. Концептуально, більшість СКВ зберігають інформацію як список файлових редагувань. Ці системи (CVS, Subversion, Perforce, Bazaar тощо) розглядають інформацію як список файлів та змін кожного з них протягом деякого часу [2]. Зображення дельта підходу до збереження файлів наведено на рисунку 1.4.

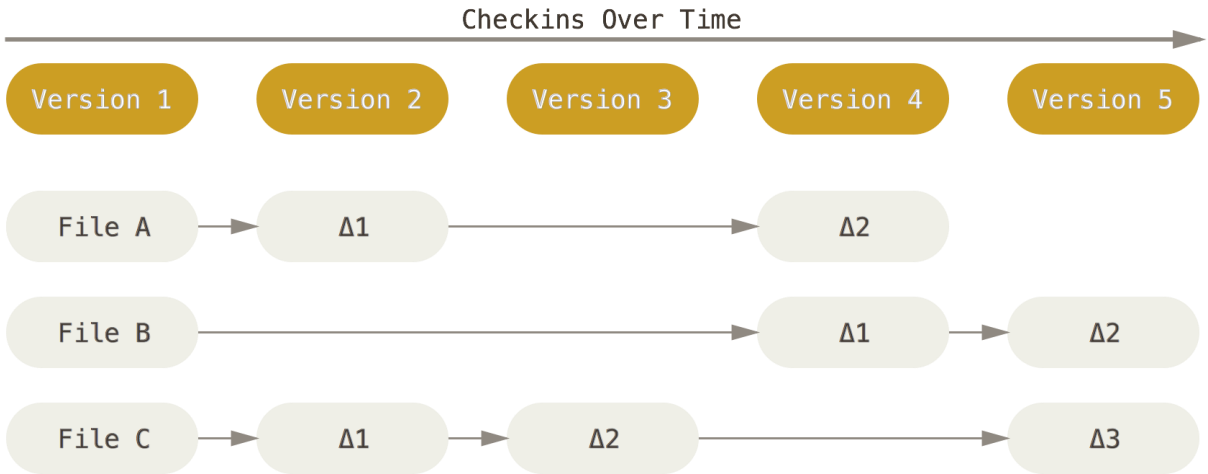

Рисунок 1.4 – Збереження даних, як переліку змін від базової версії кожного файлу

Git не оброблює та не зберігає свої дані таким чином. Замість цього, Git сприймає свої дані більш як набір знімків мініатюрної файлової системи. Кожен раз, коли робите комміт, тобто зберігаєте стан вашого проекту в Git, він запам'ятовує як виглядають всі ваші файли в той момент і зберігає посилання на цей знімок. Для ефективності, якщо файли не змінилися, Git не зберігає файли знову, просто робить посилання на попередній ідентичний файл, котрий вже зберігається. Git вважає свої дані більш як потік знімків. Зображення git підходу до збереження файлів наведено на рисунку 1.5.

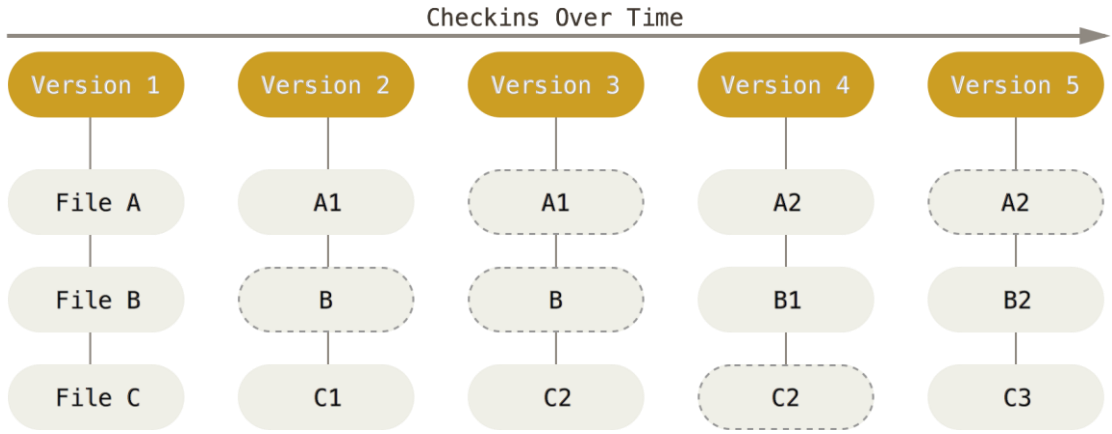

Рисунок 1.5 – Зберігання даних як знімків проекту за хронологією

Це дуже важлива різниця між Git та майже всіма іншими СКВ. З цієї причини в Git було заново переосмислено майже кожен аспект контролю версій, які інші системи просто копіювали з попереднього покоління. Це зробило Git більш схожим на мініатюрну файлову систему з деякими неймовірно потужними вбудованими інструментами на додаток, а не просто СКВ.

Більшість операцій у Git потребують лише локальних файлів та ресурсів для здійснення операцій – немає необхідності в інформації з інших комп'ютерів мережі. Якщо ви звикли до ЦСКВ, де більшість операцій обтяжені такими мережевими запитами, то цей аспект може привести до думки, що боги швидкості наділили Git неземною силою. Через те, що повна історія проекту знаходиться на вашому локальному диску, більшість операцій здійснюються майже миттєво.

Наприклад, для перегляду історії проекту Git не має потреби брати її з серверу, він просто зчитує її прямо з локальної бази даних. Це означає, що ви отримуєте історію проекту не встигнувши кліпнути оком. Якщо бажаєте переглянути відмінності між поточною версією файлу та його редакцією місячної давності, Git знайде копію збережену

місяць тому і проведе локальне обчислення різниці замість того, щоб звертатись за цим до віддаленого серверу чи спочатку робити запит на отримання старішої версії файлу.

Також це означає, що за відсутності мережевого з'єднання не будете мати особливих обмежень. Перебуваючи в літаку чи потязі можна цілком комфортно створювати коміти, доки не відновите з'єднання з мережею для їх відвантаження. Якщо ви прийшли додому та не можете змусити належним чином працювати свій VPN-клієнт, усе одно можна продовжувати роботу. У багатьох інших системах подібні дії або неможливі, або пов'язані з безліччю труднощів. Наприклад, у Perforce, без з'єднання з мережею не вдасться зробити багато; у Subversion та CVS можете редагувати файли, але не можете створювати коміти з внесених змін (оскільки немає зв'язку з базою даних).

Будь-що в Git перед збереженням отримує контрольну суму, за якою потім і можна на нього посилатися. Таким чином, неможливо змінити файл чи директорію так, щоб Git про це не дізнався. Цей функціонал вбудовано в систему на найнижчих рівнях і є невід'ємною частиною її філософії. Не можливо втратити інформацію при передачі чи отримати пошкоджений файл без відома Git.

Механізм, який використовується для цього контролю, називається хеш SHA-1. Він являє собою 40-символьну послідовність цифр та перших літер латинського алфавіту (a-f) і вираховується на основі вмісту файлу чи структури директорії в Git. Фактично, Git зберігає все не за назвою файлу, а саме за значенням хешу його змісту.

Коли виконуєте певні дії в Git, при цьому, майже завжди відбувається виключно додавання інформації до бази даних Git. Складно змусити систему зробити щось невиправне чи повністю видалити дані будь-яким чином. Як і в будь-якій СКВ, можно втратити чи зіпсувати лише не збережені в коміті зміни. Але це майже неможливо, коли вже збережено знімок, особливо, якщо регулярно надсилаєте свою базу до іншого сховища.

Git має три основних стани, в яких можуть перебувати файли: збережений у коміті (commited), змінений (modified) та індексований (staged). Збережений у коміті означає, що дані безпечно збережено в локальній базі даних. Змінений означає, що у файл внесено редагування, які ще не збережено в базі даних. Індексований стан виникає тоді, коли позначаєте змінений файл у поточній версії, щоб ці зміни ввійшли до наступного знімку коміту.

Це приводить до трьох головних відділів проекту під управлінням Git: директорія Git, робоча директорія та індекс, наведені на рисунку 1.6.

У директорії Git система зберігає метадані та базу даних об’єктів проекту. Це найважливіша частина Git, саме вона копіюється при клонуванні сховища з іншого комп’ютеру.

Індекс — це файл, що зазвичай знаходиться в директорії Git і містить інформацію про те, що буде збережено у наступному коміті. Також цей файл називають “областю додавання” (staging area).

Для кожного відстежуваного файлу Git записує таку інформацію, як розмір, час створення і час останньої зміни, у файлі, відомому як index. Щоб визначити, чи був файл змінений, Git порівнює його поточні характеристики із збереженими в індексі. Якщо вони збігаються, то Git не стане перечитувати файл заново.

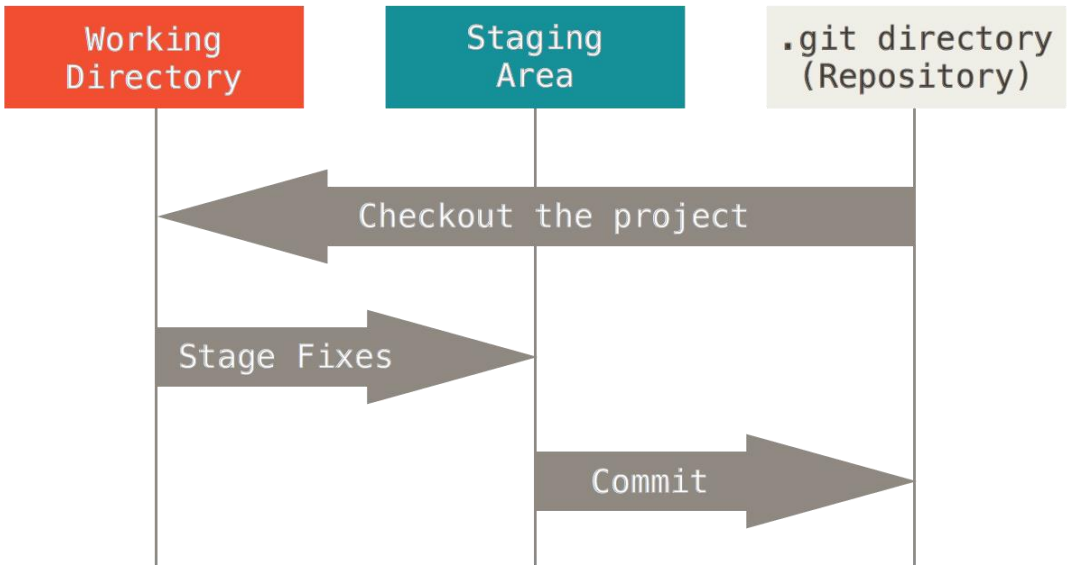

Рисунок 1.6 – Робоча директорія, індекс та директорія Git

Оскільки зчитування цієї інформації значно швидше, ніж читання всього файлу, то якщо ви редагували лише кілька файлів, Git може оновити свій індекс майже миттєво.

Найпростіший процес взаємодії з Git виглядає приблизно так:

1. Ви редагуєте файли у своїй робочій директорії.
2. Надсилаєте файли до індексу, що зберігає їхній поточний стан.
3. Створюєте фіксацію: знімок з індексу остаточно зберігається в директорії Git.

У випадку, якщо окрема версія файлу вже є в директорії Git, цей файл вважається збереженим у коміті. Якщо він зазнав змін і перебуває в індексі, то він індексований.

Якщо ж його стан відрізняється від того, який був у коміті, і файл не знаходиться в індексі, то він називається зміненим, зазначимо це на рисунку 1.7.

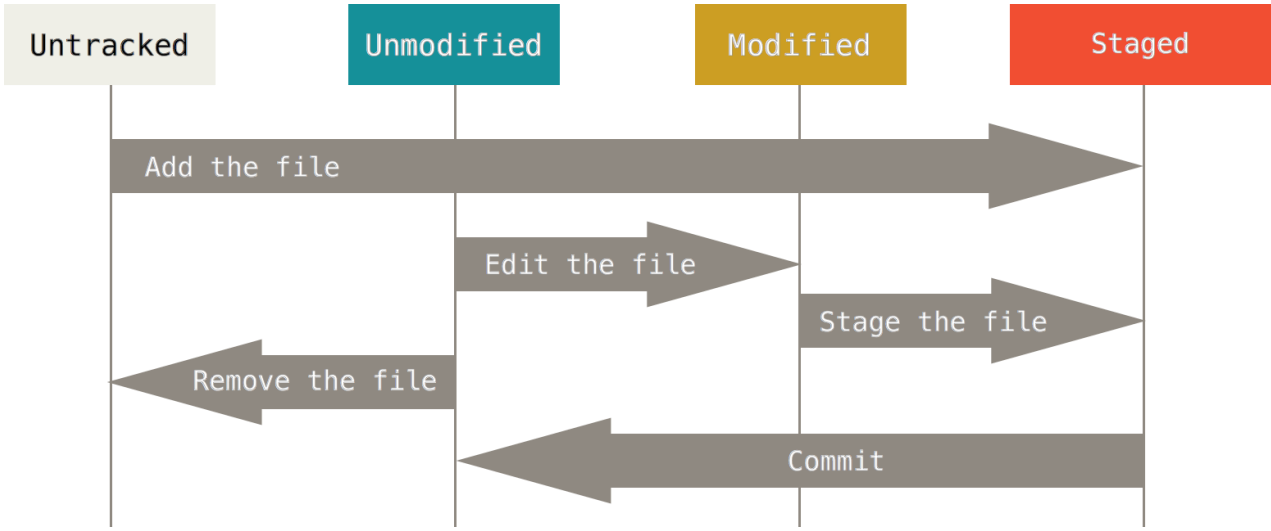

Рисунок 1.7 – Цикл життя статусу ваших файлів

1.5.2 Галуження у Git

Коли фіксуєте зміни, Git зберігає об’єкт фіксації, що містить вказівник на знімок змісту, який додано. Цей об’єкт також містить ім’я та поштову адресу автора, набране вами повідомлення та вказівники на фіксацію або фіксації, що були прямо до цієї фіксації (батько чи батьки): нуль для першої фіксації, одна фіксація для нормальної фіксації, та декілька фіксацій для фіксацій, що є результатом злиття двох чи більше гілок [2].

Щоб це уявити, припустимо, що є тека з трьома файлами, які додали та зафіксували. Додання файлів обчислює контрольну суму для кожного (SHA-1 хеш), зберігає версію файлу в сховищі Git (Git називає їх блобами) та додає їх контрольні суми до області додавання.

Коли створили фіксацію, Git обчислив контрольну суму кожної теки (у цьому випадку, тільки кореневої теки) та зберігає ці об’єкти дерева в сховищі Git. Потім Git створює об’єкт фіксації, що зберігає метадані та вказівник на корінь дерева проекту, щоб він міг відтворити цей знімок, коли потрібно.

Ваше Git сховище тепер зберігає п’ять об’єктів: по одному блобу зі змістом на кожен з трьох файлів, одне дерево, що перелічує зміст теки та вказує, які файли зберігаються у яких блобах, та одну фіксацію, що вказує на корінь дерева та зберігає

метадані фіксації. Важливо завжди розуміти, що Git працює з деревом, а це дозволяє працювати з даними значно швидше. Можемо віднести цей пункт, як одну з переваг децентралізованої системи керування версіями Git.

Також важливо ще раз зауважити, що Git обчислює хеш для кожного з файлів. Тому завжди можемо бути впевнені, що використовуємо ті файли, які бажаємо, а також маємо можливість перевірити навіть дрібніші зміни.

1.5.3 Процеси роботи Git

На відміну від централізованих систем керування версіями, розподілена сутність Git дозволяє бути набагато більш гнучким щодо того, як розробники співпрацюють над проектами. У централізованих системах кожен розробник є одним з вузлів, які працюють більш менш на рівних над централізованим розподілювачем (hub). У Git, втім, кожен розробник є потенційно і вузлом, і сервером – тобто, кожен розробник може як надсилати код до інших репозиторіїв, так і супроводжувати публічний репозиторій, на якому інші можуть базувати свою роботу, та до якого вони можуть надсилати зміни. Отже розглянемо декілька поширених парадигм [2].

У централізованих системах, взагалі існує єдина модель співпраці – централізований процес роботи. Один центральний розподілювач або репозиторій може приймати код, і всі синхронізують свою роботу з ним. Усі розробники є вузлами – споживачами цього розподілювача – та синхронізуються виключно з ним. Це означає, що, якщо два розробники зроблять клон з розподілювача та обидва щось змінять, перший, хто надішле свої зміни назад, зможе це зробити без проблем. Другий розробник змушений зливати роботу першого до надсилання змін, щоб не переписати зміни першого розробника, ця схема наведена на рисунку 1.8.

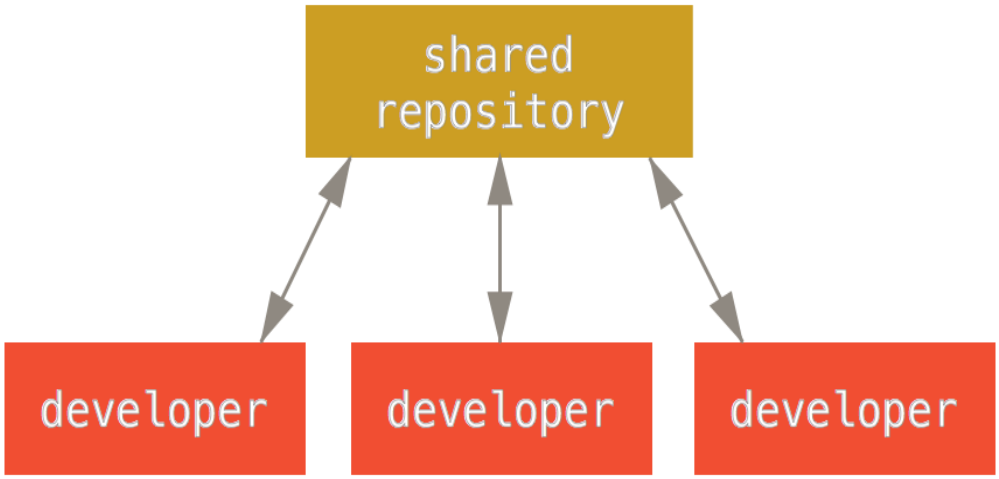

Рисунок 1.8 – Централізований порядок роботи

Оскільки Git дозволяє мати декілька віддалених сховищ, можливо мати процес роботи, в якому кожен розробник має право на запис до власного публічного репозиторію та доступ на читання з репозиторіїв інших розробників. Цей сценарій зазвичай включає канонічне сховище, яке представляє “офіційний” проект. Щоб зробити внесок до такого проекту, треба створити власний публічний клон проекту та надіслати зміни до нього. Потім, можете надіслати запит до супроводжувача головного проекту, щоб він злив ваші зміни. Супроводжувач тоді може додати ваше сховище як віддалене, перевірити ваші зміни локально, злити їх до своєї гілки та надіслати до свого репозиторію. Процес зображено на рисунку 1.9:

1. Супроводжувач проекту надсилає зміни до свого власного репозиторію.
2. Розробник клонує цей репозиторій та робить зміни.
3. Розробник надсилає зміни до своєї власної публічної копії.
4. Розробник надсилає супроводжувачу листа, в якому просить злити його зміни.
5. Супроводжувач додає сховище розробника як віддалене та зливає зміни локально.
6. Супроводжувач надсилає злиті зміни до головного репозиторію.

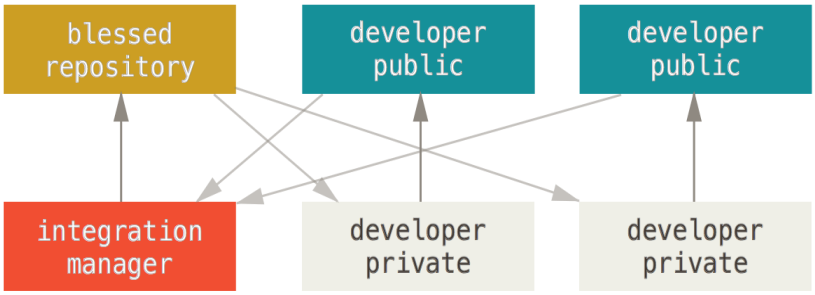

Рисунок 1.9 – Процес роботи з менеджером інтеграції

Процес роботи з диктатором та лейтенантами – це варіація процесу роботи з багатьма репозиторіями. Зазвичай він використовується у величезних проектах з сотнями учасників. Різноманітні менеджери інтеграції відповідають за окремі частини репозиторію - їх називають лейтенантами. Всі лейтенанти мають одного менеджера інтеграції, відомого як доброзичливий диктатор. Репозиторій доброзичливого диктатора слугує зразковим сховищем, з якого всі розробники мають отримувати зміни. Процес працює наступним чином (рис. 1.10):

1. Звичайні розробники працюють у власній тематичній гілці та перебазовують свою роботу поверх master. Гілка master та, що в диктатора.
2. Лейтенанти зливають тематичні гілки розробників до власних гілок master.
3. Диктатор зливає гілки лейтенантів master до гілки диктатора master.
4. Диктатор надсилає гілку master до зразкового репозиторію, щоб інші розробники могли перебазуватись поверху неї.

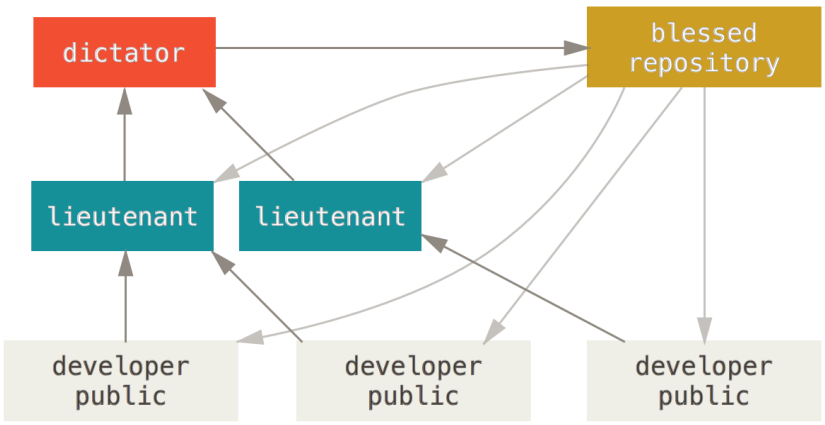

Рисунок 1.10 – Процес роботи з доброзичливим диктатором

Такий процес роботи не є поширеним, проте може бути корисним у дуже великих проектах або в дуже ієрархічних середовищах. Він дозволяє керівнику проекту (диктатору) делегувати чимало роботи та збирати великі частини коду з декількох місць перед тим, як інтегрувати їх.

1.6 Постановка задачі

Метою роботи є дослідження методів обробки та сегментації відео та розробка системи для збереження та кодування мультимедійного проекту, керування їм.

Для цього будемо використовувати кластер з серверів, що надасть змогу використовувати наше обладнання 100% часу. Це, в свою чергу, надасть можливість обробляти відео значно швидше за допомогою того, що один файл буде оброблятися одночасно на багатьох машинах.

Об`єктом дослідження є можливість вирішення проблеми швидкісної обробки відео.

Предмет дослідження – методи розподіленої обробки відеоданих.

2 РОЗПОДІЛЕНА ОБРОБКА ВІДЕО

2.1 Розподілена обробка даних

MapReduce - це модель програмування та відповідна реалізація для обробки великих наборів даних. Користувачі задають функцію Map, яка обробляє пару ключ/значення, щоб генерувати набір проміжних пар ключ/значення та функцію Reduce, яка об'єднує всі проміжні значення, пов'язані з тим самим проміжним ключем. Багато реальних завдань є вираженими в цій моделі. Програми, написані в цьому функціональному стилі, автоматично розпаралелюються і виконуються на великому кластері машин. Система дбає про подробиці розподілу вхідних даних, планування виконання програми через набір машин, обробку помилок машини та управління необхідним міжмережевим зв'язком. Це дозволяє програмістам без досвіду роботи з паралельними та розподіленими системами легко використовувати ресурси великої розподіленої системи. Розглянемо цю модель на прикладі реалізації компанії Google.

За останні п'ять років працівники Google реалізували сотні спеціальних формул обчислення, які обробляють великі обсяги вихідних даних, таких як скановані документи, журнали веб-запитів тощо, також обчислення різноманітних видів похідних даних, наприклад, перевернуті індекси, різні представлення структури графа веб-документів, резюме кількості сторінок сканування на хост, набір найбільш частих запитів у даний день, тощо. Більшість таких обчислень концептуально прості, однак вхідні дані зазвичай великі, а розрахунки повинні бути розподілені по сотням або тисячам машин, щоб закінчити це за розумний час. Проблеми того, як паралелізувати обчислення, розподіл даних та їх обробку можуть змушувати використовувати оригінальні прості обчислення з великою кількістю складного коду. Як реакція на цю проблему Google спроектували нову абстракцію, яка дозволяє виразити прості обчислення, які ми намагаємося виконати, але приховуючі брудні деталі розпаралелювання, відмовостійкісті, розподілу даних та балансування навантаження у кластері. Google зрозуміли, що більшість наших обчислень включає застосування операцій Map до кожного логічного "запису" у нашому вхідному потоку даних для того, щоб обчислити набір проміжних пар ключ/значення, а потім, застосовуючи скорочену операцію до всіх значень, якими поділилися той самий ключ, щоб правильно поєднати отримані дані. Їх використання функціональною моделю з невизначеною функцією Map та Reduce дозволяє паралелізувати великі обчислення легко

та використовувати повторне виконання як основний механізм відмовостійкості. Усе це дає простий та потужний інтерфейс, що дозволяє автоматично розпаралелювати і розподіляти великомасштабні обчислення, об'єднані за допомогою цього інтерфейсу, який досягає високої продуктивності на великих кластерах машин.

2.1.1 Програмна модель

Обчислення приймає набір вхідних пар ключ/значення і створює набір вихідних пар ключ/значення. Користувач бібліотеки MapReduce виражає розрахунок як дві функції: Map і Reduce.

Функція Map бере пару вхідних даних і створює набір проміжних пар ключ/значення. Бібліотека об'єднує всі проміжні значення, пов'язані з тим самим проміжним ключем I, і передає їх до функції Reduce.

Функція Reduce приймає проміжний ключ I і набір значень для цього ключа. Це об'єднує разом ці значення, щоб сформувати можливо менший набір значень. Зазвичай результатом цієї функції буде один чи жодного виводу для кожного виклику. Проміжні значення подаються на функцію зменшення користувача за допомогою ітератора. Це дозволяє обробляти списки значень, які занадто великі для розміщення в пам'яті.

2.1.2 Опис реалізації

Можливі різні реалізації інтерфейсу MapReduce. Правильний вибір залежить від навколишнього середовища. Наприклад, одна реалізація може бути придатною для невеликої машини спільного користування, інша - для великого багатопроцесорного NUMA, а ще інша - для ще більшого набору мережевих машин. Для прикладу розглянемо найбільш розповсюджену конфігурацію у Google - великі кластери комп’ютерів, з'єднані з мережею, більш конкретно:

– Машини - це, як правило, двопроцесорні комп’ютери x86, що працюють під керуванням Linux, з 2-4 ГБ пам'яті на кожну машину.

– Використовується апаратно-програмна мережева апаратура - як правило, 100 мегабіт на секунду або 1 гігабіт на секунду на рівні апарата, але в цілому загальна пропускна здатність кожної секції значно менша.

– Кластер складається з сотень або тисяч машин, і тому помилки будуть загальні.

– Зберігання забезпечується за допомогою дешевих дисків IDE, безпосередньо спрямованих на окремі машини. Розроблена власна розподілена файлова система використовується для управління даними, що зберігаються на цих дисках. Файлова система використовує реплікацію, щоб забезпечити доступність та надійність на вершині ненадійного апаратного забезпечення.

– Користувачі подають задачі до системи планування. Кожна задача складається з безлічі завдань і відображається планувальником до набору доступних машин у кластері.

2.1.3 Опис виконання

Map виклики розподіляються по декількох машинах шляхом автоматичного розділення вхідних даних на набір M розділів. Розділи на вхід можна паралельно обробляти різними машинами. Зниження кількості викликів розподіляється шляхом розбиття проміжного простору ключів на частини R, використовуючи наприклад таку функцію розділення (2.1)

$$hash(key) \bmod R \qquad (2.1)$$

Кількість розділів (R) та функція розбиття задається користувачем. На рисунку 2.1 показано загальний потік роботи MapReduce.

Коли програма викликає функцію MapReduce, відбувається наступна послідовність дій (нумеровані мітки на рисунку 1 відповідають номерам у списку нижче):

1. Бібліотека MapReduce в програмі користувача спочатку розбиває вхідні файли на M частинки, як правило, від 16 мегабайт до 64 мегабайт на шматки. Наступним кроком він запускає багато копій програми на кластері машин.

2. Одна з примірників програми особлива - майстер. Решта - це працівники, яким призначено роботу майстра. Є M map завдань, та R reduce завдань, які потрібно призначити. Майстер вибирає працездатних працівників і призначає кожному задачу map або reduce.

3. Працівник, якому присвоєно завдання map, читає вміст відповідного вводу. Він аналізує пари ключ / значення і передає кожну пару користувальницькій функції map. Міжмірні пари ключів / значень, вироблені за допомогою функції «Карта», буферизовані в пам'яті.

4. Періодично буферизовані пари записуються на локальний диск, розділені на R-ділянки за допомогою функції розділення. Розташування цих буферних пар на локальному диску повертається майстру, який несе відповідальність за пересилання цих місцеположень до reduce працівників.

5. Коли reduce працівник отримує локацію від майстра, він використовує віддалені виклики для того, щоб прочитати буфер за адресом, який йому дав майстер. Коли reduce працівник прочитав усі внутрішні дані, він сортує їх проміжними ключами, щоб усі входи одного і того ж ключа були згруповані разом. Сортування потрібне, тому що зазвичай багато різних ключів прямують до однієї функції скорочення.

6. Reduce функція повторюється над сортованими проміжними даними та для кожного унікального проміжного ключа, та відповідний набор проміжних значень передається до reduce функції користувача. Вихід функції reduce додається до кінцевого вихідного файлу для цього reduce розділу.

7. Після завершення всіх map і reduce завдань майстер запускає користувацьку програму. На цьому етапі виклик MapReduce у користувацькій програмі повертається назад до коду користувача.

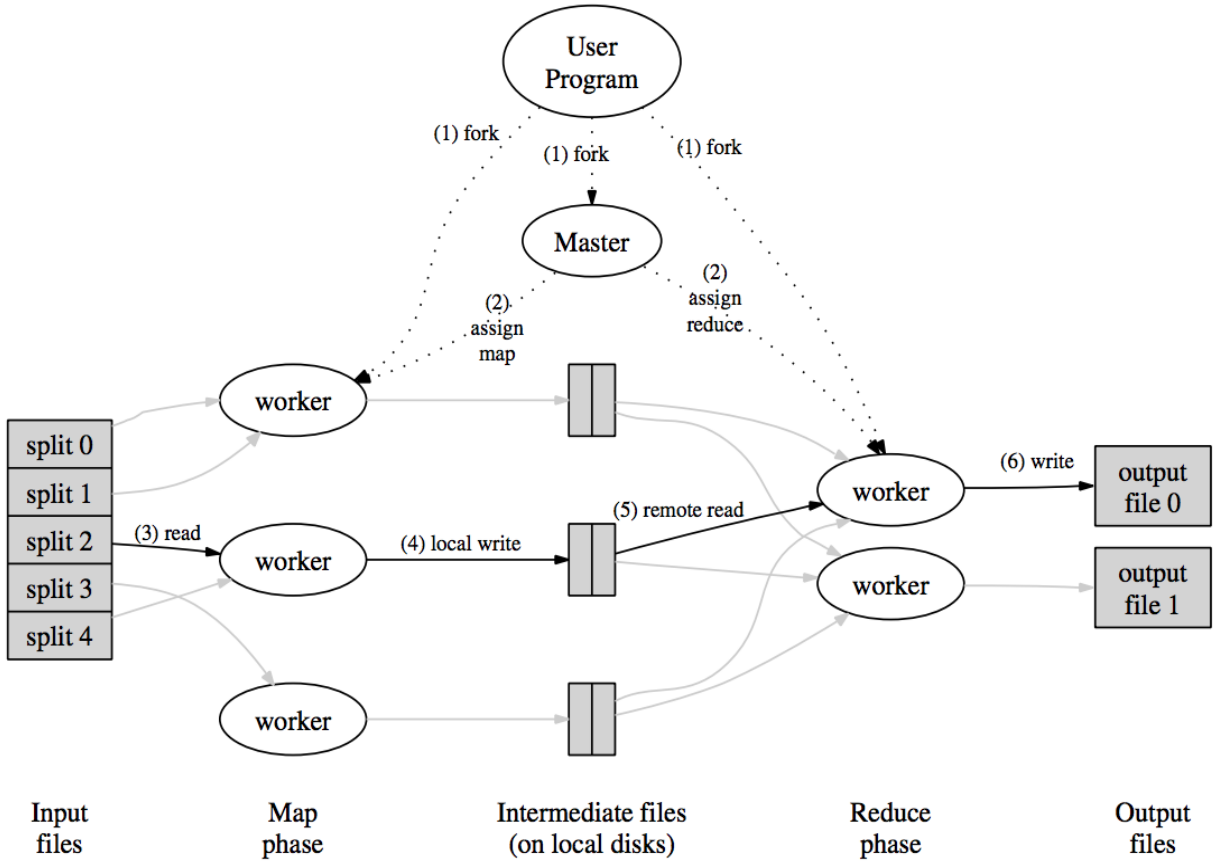

Рисунок 2.1 – Загальний потік роботи MapReduce

Після успішного завершення висновок виконання MapReduce доступний у R вихідних файлах (по одному на кожну reduce задачу, з іменами файлів, визначеними користувачем). Як правило, користувачам не потрібно об'єднувати ці вихідні файли R в один файл - вони часто передають ці файли як вхідні до іншого виклику MapReduce або використовують їх з іншої розповсюдженої програми, яка здатна обробляти дані, розділені на декілька файлів.

2.1.4 Структура даних майстра

Майстер зберігає кілька структур даних. Для кожної map та reduce завдання зберігається стан (незайняте, у процесі або завершено), а також ідентифікація робочої машини (для незайнятих завдань).

Майстер - це зв’язуюче ціле між map та reduce задачами, який робить це використовуючи дані про розташування регіонів файлів, у які записують map задачі, а

потім будуть читати reduce задачі. Тому для кожного завершеного завдання map майстер зберігає розташування та розміри R-інтервальних областей файлів, вироблених map завданням. Оновлення цієї інформації щодо розміщення та розміру отримуються, оскільки завдання map завершено. Поступово інформація поступово надається reduce працівникам.

2.1.4 Обробка помилок

Оскільки бібліотека MapReduce призначена для обробки дуже великої кількості даних із використанням сотень або тисяч машин, бібліотека повинна грамотно обробляти помилки машини.

Майстер періодично пильнує кожного працівника. Якщо працівник не дає відповідь протягом певного періоду часу, майстер відмічає робочого як зламаного. Будь-які завдання map, виконані працівником, скидаються до початкового режиму очікування і, таким чином, їх оброблять інші працівники. Аналогічним чином, будь-яке завдання map або reduce при виконанні з такої машини також скидається у стан перепланування.

Завершені завдання map повторно виконуються при відмові, тому що їх вихід зберігається на локальному диску зламаного апарата і тому недоступний. Завершені reduce завдання не потрібно повторно виконувати, оскільки їх вихід зберігається у глобальній файловій системі.

Коли завдання map виконується спочатку працівником А, а пізніше виконується працівником В (оскільки А не вдалося), усі працівники, що виконують reduce завдання, повідомляються про повторне виконання. Будь-яке reduce завдання, яке ще не читало дані працівника А, буде читати дані працівника Б.

MapReduce є стійким до масштабних невдач працівників. Наприклад, під час однієї операції MapReduce підтримка мережі на робочому кластері призвела до того, що групи з 80 машин за раз стали недосяжними протягом декількох хвилин. Майстер MapReduce просто повторно виконав роботу, виконану недоступними робочими машинами, і продовжував робити прогрес, в результаті завершивши операцію MapReduce.

Якщо буде проблема з майстром, нова копія почне роботу.

Коли надані користувачем map та reduce функції зменшують оператори, вони володіють детерміністичними функціями їх вхідних значень, розподілена реалізація виробляє той самий вихід, який був би зроблений неспроможним послідовним виконанням всієї програми.

Кожне завдання, що виконується, записує вихідний файл до приватних тимчасових файлів. Reduce завдання видає один такий файл, і завдання карти видає R таких файлів (по одному на кожну reduce задачу). Коли завдання map завершується, працівник надсилає повідомлення майстру і включає в них імена тимчасових файлів R. Якщо майстер отримує повідомлення про завершення вже виконаного завдання map, він ігнорує повідомлення. В іншому випадку він записує імена R-файлів у структурі основних даних.

Після завершення завдання reduce працівник автоматично перейменовує тимчасовий вихідний файл у остаточний вихідний файл. Якщо одне і те ж reduce завдання виконується на множині машин, декілька викликів перейменування будуть виконуватися для одного і того ж кінцевого вихідного файлу.

Переважна більшість map та reduce операторів є детерміністичними, а той факт, що наша семантика еквівалентна послідовному виконанню в цьому випадку робить це дуже простим для програмістів для зрозуміння поведінки їх програм. Коли map та/або reduce не є детермінованими ми забезпечуємо слабку, але все-таки розумну семантику. За наявності недетермінованих операторів вихід певної задачі reduce R1 еквівалентний виводу для R1, який виробляється послідовним виконанням недетерміністичної програми. Проте вихід для іншої задачі reduce R2 може відповідати виводу для R2, який виникає при іншому послідовному виконанні недетермінованої програми.

2.1.5 Складність завдань

Ми підрозділяємо map етап на M-шматки та reduce этап на R-шматки, як описано вище. В ідеалі, M і R повинні бути набагато більше, ніж кількість робочих машин. Кожен працівник виконує багато різних завдань, покращує динамічне навантаження, а також прискорює відновлення, коли працівник зазнає невдачі: багато завдань map, які він завершив, можна розподілити на всіх інших робочих машинах.

Є практичні межі того, наскільки великі M і R можуть бути в нашій реалізації, оскільки майстер повинен приймати рішення про планування (2.1) і зберігати стан (2.2) O (M * R) в пам'яті, як описано вище.

$$O(M + R) \tag{2.2}$$

$$O(M * R) \tag{2.3}$$

Постійні фактори використання пам'яті невеликі, однак: частина (2.2) стану складається з приблизно одного байту даних на map завдання/reduce пари задач. Крім того, R часто обмежується користувачами, оскільки випуск кожної задачі зменшення закінчується в окремому файлі виходу. На практиці ми схильні вибирати M так, щоб кожне окреме завдання становило приблизно від 16 Мб до 64 Мб вхідних даних (так що оптимізація місцевості, описана вище, є найбільш ефективною), і ми робимо R невеликим краєм числа працівників машини, які очікуємо використовувати. Ми часто виконуємо обчислення MapReduce з M = 200 000 і R = 5 000, використовуючи 2000 робочих машин.

2.1.6 Результати тестування моделі

Модель проста у використанні навіть для програмістів, які не мають досвіду роботи з паралельними та розподіленими системами, оскільки приховує подробиці розпаралелювання, відмовостійкісті, оптимізацію локалізації та балансування навантаження. Реалізація ефективно використовує машинні ресурси і тому підходить для використання на багатьох великих обчислювальних задачах.

Обмеження моделі програмування дозволяє легко виразити і розповсюджувати обчислення, а також робити такі обчислення виправданими. Смуга пропускання мережі є дефіцитним ресурсом. Тому низка оптимізація в системі спрямовані на зменшення кількості даних, що передаються по мережі: оптимізація локального середовища дає нам змогу читати дані з локальних дисків, а також запис однієї копії проміжних даних на локальний диск, зберігаючи пропускну здатність мережі. По-третє, надмірне виконання може бути використане для зменшення впливу повільних машин, а також для усунення несправностей машини та втрати даних.

2.2 Кодування відео за допомогою стандарту HEVC

HEVC - формат відео стиснення із застосуванням ефективніших алгоритмів у порівнянні з H.264/MPEG-4 AVC. Рекомендація стандарту розроблена в зв'язку зі зростаючою потребою в більш високому ступені стиснення рухомих зображень для самих

різних додатків, таких як потокова передача в Інтернеті, передача даних, відеоконференц-зв'язок, цифрові пристрої, що запам'ятовують і телевізійне мовлення.

2.2.1 Структура HEVC

При кодуванні відео в HEVC застосовується такий же «гібридний» підхід, що і у всіх сучасних кодеків, починаючи з H.261. Схема кодера HEVC зображена на рисунку 2.2.

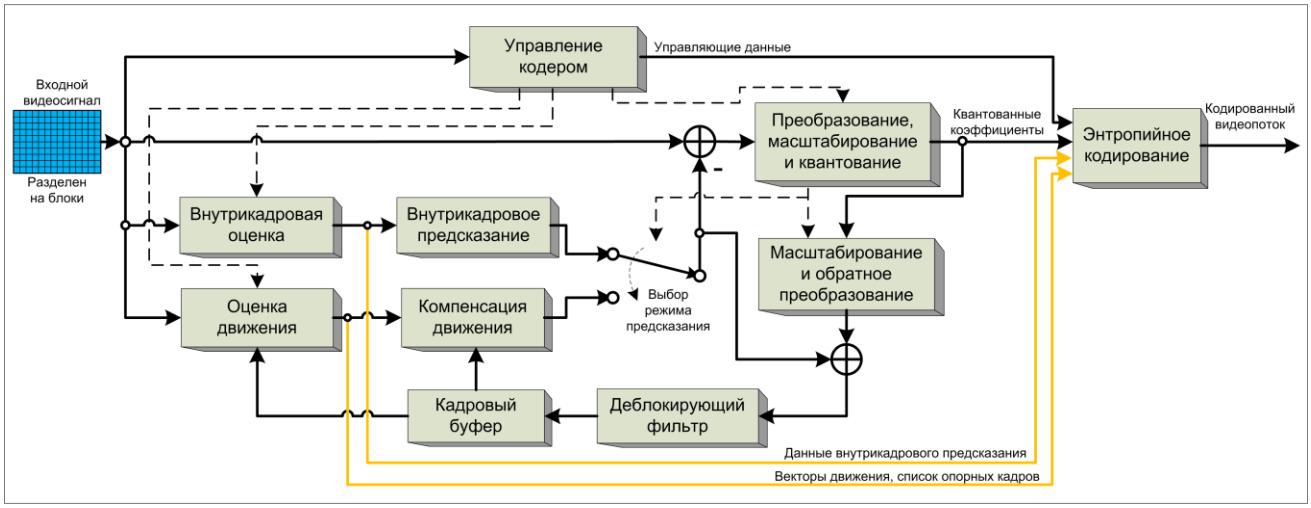

Рисунок 2.2 – Схема кодера HEVC

У кодері HEVC кожен відеокадр ділиться на блоки. Перший кадр відеопослідовності кодується з використанням тільки внутрикадрового передбачення, тобто застосовується просторове передбачення очікуваного рівня відліку всередині кадру по сусіднім відлікам, при цьому відсутня залежність від інших кадрів. Для більшості блоків всіх інших кадрів послідовності, як правило, використовується режим міжкадрового тимчасового передбачення. У режимі міжкадрового передбачення на підставі даних про величину відліків опорного кадру та вектора руху оцінюються поточні відліки кожного блоку. Кодер і декодер створюють ідентичні міжкадрові передбачення шляхом застосування алгоритму компенсації руху за допомогою векторів руху і даних обраного режиму, які передаються в якості додаткової інформації.

Різницевий сигнал передбачення, який представляє собою різницю між опорним блоком кадру і його пророкуванням, піддається лінійному просоровому перетворенню.

Далі коефіцієнти перетворення масштабуються, квантуються, застосовується ентропійне кодування та передаються разом з інформацією передбачення.

Кодер в точності повторює цикл обробки декодером так, що в обох випадках будуть генеруватися ідентичні передбачення наступних даних. Таким чином, перетворені квантовані коефіцієнти піддаються зворотному масштабуванню і потім зворотному перетворенню, щоб декодувати значення різницевого сигналу. Різниця потім додається до передбачення, і отриманий результат фільтрується для згладжування артефактів, отриманих поділом на блоки і при квантуванні. Остаточне уявлення кадру (ідентичне кадру на виході декодера) зберігається в буфері декодованих кадрів, яке буде використовуватися для прогнозування подальших кадрів. У підсумку, порядок кодування і декодування обробки кадрів часто відрізняється від порядку, в якому вони надходять з джерела.

Передбачається, що відеоматеріал на вході кодера HEVC має прогресивну розгортку. У HEVC не представлено явних функцій кодування чергуванням розгортки, так як строкова розгортка не використовується в сучасних дисплеях і має все менше поширення. Проте, в HEVC були представлені метадані, що дозволяють вказати кодеру, що було закодовано відео з чергуванням розгортку в одному з двох режимів: у вигляді окремих зображень, як два поля (парні або непарні рядки кадру) або весь кадр цілком. Цей ефективний метод забезпечує кодування відеосигналу з чергуванням рядків, минаючи необхідність навантажувати декодери підтримкою спеціального процесу декодування.

Розробка більшості стандартів відеокодування призначена, в першу чергу, для досягнення найбільшої ефективності кодування. Ефективність кодування визначається здатністю закодувати відео з мінімально можливою швидкістю передачі даних при збереженні певного рівня якості відео. Існує два стандартних способу вимірювання ефективності кодування відео, один з яких полягає в використанні об'єктивної метрики, такий як пікового відношення сигнал-шум (PSNR), а другий полягає у використанні суб'єктивної оцінки якості відео. Суб'єктивна оцінка якості зображення є найбільш важливим параметром для оцінки кодування відео, так як глядачі сприймають якість відео саме суб'єктивно.

2.2.2 Ефективність HEVC

Замість макроблоків, які застосовувалися в H.264, в HEVC використовуються блоки з деревовидної структурою кодування. Виграш кодера HEVC - в застосуванні блоків більшого розміру. Це було показано в тестах PSNR, де порівнювалися результати кодування з різними розмірами блоків.

В результаті тестів було показано, що в порівнянні з кодуванням блоків розміром 64x64 пікселів, бітрейт збільшується на 2,2%, коли використовуються блоки розміром 32x32 і збільшується на 11,0%, коли використовується розмір блоків 16x16. У тестах кодування відео з роздільною здатністю 2560x1600 пікселів при використанні блоків з розміром 32x32 пікселів бітрейт збільшується на 5,7%, а при використанні блоків розміром 16x16 пікселів - на 28,2%, в порівнянні з відео, де використані блоки розміром 64x64 при однаковому піковому відношенні сигнал-шум.

Тести показали, що застосування блоків більшого розміру більш ефективно при кодуванні відео з високою роздільною здатністю. Тести також показали, що для декодування відео, закодованого з розмірами блоків 16x16, потрібно на 60% більше часу, ніж при використанні блоків 64x64. Тобто, застосування блоків великих розмірів підвищує ефективність кодування при одночасному скороченні часу декодування.

Було проведено порівняння ефективності кодування основного профілю H.265 з кодеками H.264 / MPEG-4 AVC High Profile (HP), MPEG-4 Advanced Simple Profile (ASP), H.263 High Profile Latency (HLP) і H .262 / MPEG-2 Main Profile (MP). Були закодовані відео розважальних програм і дев'ять тестових відеопослідовностей з дванадцятьма різними бітрейтами з використанням даної моделі HEVC HM-8.0, п'ять з них були з HD роздільною здатністю, а чотири були з роздільною здатністю WVGA (800 × 480). Зменшення бітрейта визначалося на основі PSNR (табл.2.1).

Таблиця 2.1 - Порівняння стандартів відеокодування при рівному PSNR

Стандарт відеокодування	Середнє скорочення бітрейта			
	H.264/MPEG-4 AVC HP	MPEG-4 ASP	H.263 HLP	H.262/MPEG-2 MP
HEVC MP	35,4 %	63,7 %	65,1 %	70,8 %
H.264/MPEG-4 AVC HP	-	44,5 %	46,6 %	55,4 %
MPEG-4 ASP	-	-	3,9 %	19,7 %
H.263 HLP	-	-	-	16,2 %

3 РОЗРОБКА СИСТЕМИ ЗБЕРЕЖЕННЯ ТА ОБРОБКИ ВІДЕО

3.1 Опис програмної моделі

Зараз слід зазначити підхід, за допомогою котрого буде реалізована мета цієї роботи. Насамперед зазначимо головні функції цієї системи:

– збереження історії перетворень усього медіа проекту;

– фіксація локальної копії для подальшого повернення до неї у будь-який час;

– підтримка гілок;

– перегляд змін проекту у зручному форматі. Якщо для змін у тестових файлах нам достатньо бачити у яких рядках зроблено зміни, і зазвичай бачимо зміну кожної строки як видалення строки з попередньої версії та додавання строки з наступної, але для медіа форматів цей підхід не є правильним, так як медіа формати, а у першу чергу відео, набагато складніші та комплексні, тому вважаю, що слід дуже уважно поставитися до цього питання. Також дуже важливо пам’ятати, що для нас більш важливо підтримувати це на етапі розробки проекту, монтажу фільму, тощо. Тому ця частина є однією з найскладніших;

– об’єднання гілок потрібно підтримати на певному рівні. Також слід зазначити, що дуже складною гілкою розробки є підтримка автоматичного розрішення конфліктів, тому що це набагато складніше, ніж така саме задача для тексту, та мабуть слід зовсім відмовитись від цього, і ця операція здійснялась лише працівниками цього проекту.

Другою частиною роботи буде створення системи, яка буде працювати окремо, і вирішати задачі, направлені на створення проекту-результату. Вважаю, що дуже важливо схематично відтворити увесь процес за допомогою схем.

Спочатку зазначимо більш загальні компоненти та фази. Схема зазначена на рисунку 3.1.

Рисунок 3.1 – Процес роботи команди з системою

Як бачимо, загальна ідея доволі проста - є команда, яка працює над проектом, та робить якісь зміни у ньому, після цього, коли команда закінчує смислову частину проекту, ця частина спрямовується до СКВ, яка гарантує надійність цих даних, та можемо бути впевненими, що зможемо повернутися до цих змін пізніше. Також СКВ спрощує роботу над проектами.

Після того, як СКВ прийняв наші зміни, проект готовий щодо його обробки. Тобто наступним кроком буде його обробка. Під обробкою можемо розуміти наприклад рендерінг відео. Для цієї мети необхідно налаштувати кластер з серверів, які будуть опрацьовувати відповідні дані.

Якщо говоримо про кластер, то треба обміркувати метод, за яким будемо його налаштовувати та оновлювати. Пропоную для цієї мети використовувати існуючу систему Docker. Яка пропонує зручний спосіб налаштування усього програмного забезпечення.

Для цього Docker використовує технологію, яка створює віртуальний контейнер, та потім він буде виконуватися на серверному обладнанні. При цьому ці контейнери будуть дуже зручні у налаштуванні та у виконанні. Як особливість можно зазначити те, що налаштовуємо ці “контейнери” як віртуальні машини, але при цьому вони будуть використовувати реальні ресурси замість віртуальних. Також, роблячи дослідження, з’ясували, що різниця між обробкою з системою, яка розгорнута на реальному обладнанні, та системою, яка працює у “контейнері” така несуттєва, що можемо вважати, що її немає.

Робота по розробці контейнеру також зручна у порівнянні з альтернативними методами.

3.2 Система збереження даних

Визначимо частини, з яких складається система збереження даних (СКВ). А саме – модуль інтерфейсу, модуль взаємодії, контролюючий модуль та модуль збереження.

Для цього спочатку намалюємо схему системи (рисунок 3.2), використовуючи цю схему, зможемо спроектувати усі її компоненти окремо та також вести розробку цих компонентів також окремо, завдяки чому кожна частина проекту є незалежною. Це спрощує розробку і тестування системи. Також у випадку непередбачуваного збою лишимось тільки одного компоненту системи, залишаючи інші без змін. Тому система буде більш стабільною.

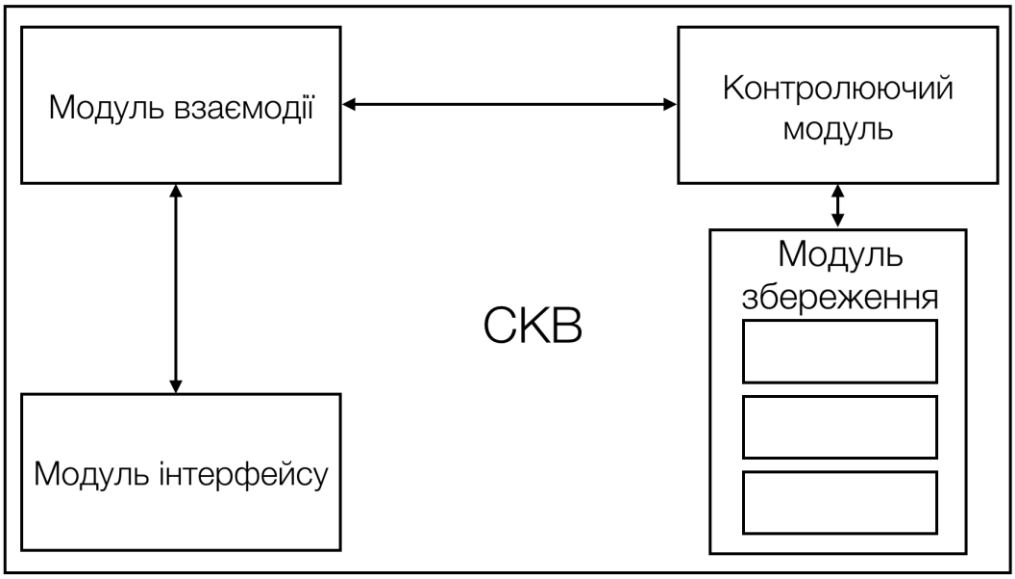

Рисунок 3.2 – Схема системи керування версіями

Далі розглянемо кожен компонент більш детально.

Модуль інтерфейсу – головний модуль, відповідний за зв’язок з клієнтом. Усі запити до системи взагалі будуть виконуватися через цей модуль. У цей модуль входить клієнтська частина (яка буде розташовуватися на машинах користувачів), та також до нього входить модуль сервера, який представляє загальний веб-інтерфейс доступу до серверу. Крім цього модулю немає таких функцій, які виконують взаємодію з клієнтом. Відповідь з системи обробки версій та її налаштування також проходить через модуль інтерфейсу. Схему модулю інтерфейсу надано на рисунку 3.3.

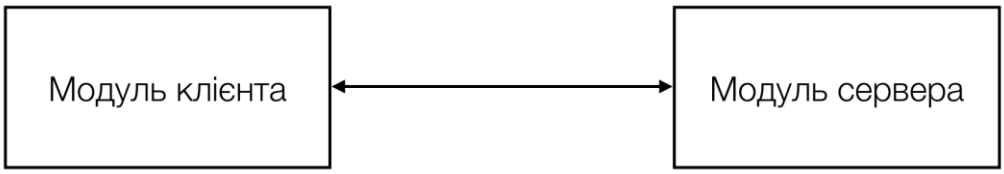

Рисунок 3.3 – Схема модулю інтерфейсу

Модуль клієнта використовується для зв’язку комп’ютера користувача з системою. Цей модуль буде відповідати за зв’язок файлів проекту на комп’ютері та на сервері, виконувати роботу їх загрузки на сервер та їх загрузки з нього. Також він буде виконувати роботу, яка потрібна для об’єднання змін з різних гілок. Взагалі можемо сказати, що це модуль, який відповідний за роботу команди з СКВ. Цей модуль буде мати пакети для всіх операційних систем.

Модуль серверу використовується системою для обробки інформації з модулю клієнту. Також будемо використовувати його для подальшої взаємодії користувача з системою. Для цього не потрібно встановлювати програмне забезпечення, так як цей модуль працює на веб-сервері, і можемо робити налаштування через браузер. Також цей модуль посилає інформацію до модулю взаємодії, та завдяки цьому інформація з комп'ютерів користувачів надходить до модулю збереження.

Далі зобразимо увесь компонент модулю сервера (рис. 3.4).

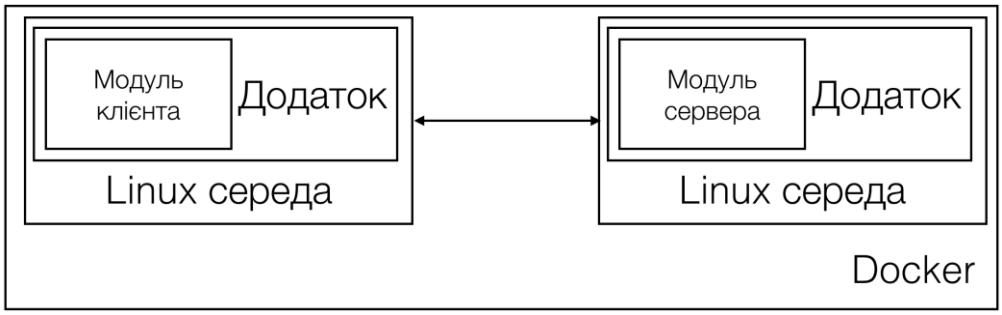

Рисунок 3.4 – Docker схема модулю сервера

Модуль взаємодії відповідає за зв'язок модулю клієнта з контролюючим модулем, а також з СОВ. Цей модуль дуже важливий у цій системі, тому що він є сполучною ланкою усієї СКВ. Без цього модулю буде неможливим збереження та загрузка даних з модулю збереження.

Далі зобразимо увесь компонент модулю взаємодії (рис. 3.5).

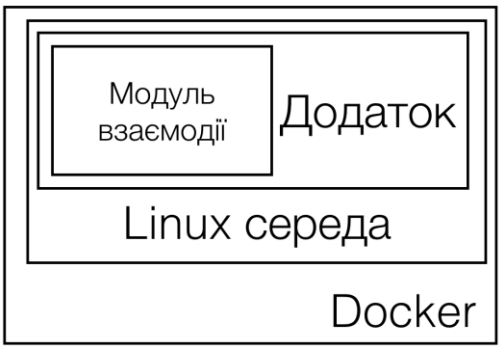

Рисунок 3.5 – Docker схема модулю взаємодії

Контролюючий модуль – проміжна ланка між системою та модулем збереження. Він потрібен для того, щоб бути впевненими, що інші модулі не будуть мати прямого зв'язку з модулем збереження та будуть працювати лише через контролюючий модуль.

Контролюючий модуль складається з модулю обробки та модулю кодування. Схема контролюючого модулю надана на рисунку 3.6.

Рисунок 3.6 – Схема контролюючого модулю

Модуль обробки – головний модуль контролюючого модулю. Він виконує декілька важливих функцій, без яких функціонування системи неможливо.

Цей модуль відповідає за зв’язок між модулем взаємодії усієї СКВ та подальшою обробкою інформації для збереження.

Після того, як інформація буде прийнята модулем обробки, він відсилає її на кодування та результат кодування зберігає.

Якщо необхідно зробити зворотну операцію, то він спочатку получає данні з модулю збереження та потім декодує їх, використовуючи модуль кодування. Результат цієї операції буде надісланий до модулю взаємодії СКВ.

Модуль кодування виконує лише дві функції:

– Кодування інформації, яка поступає з модулю обробки. Ця операція дозволить зберігати інформацію, використовуючи ресурси більш ефективно, та також це дозволить зберігати інформацію у кодованому, захищеному вигляді.

– Подальше декодування цих даних для використовування у інших модулях.

Увесь компонент контролюючого модулю представлений на рисунку 3.7.

Модуль збереження – дуже примітивний модуль, який відповідає тільки за збереження інформації на сервері та подальшу загрузку цієї інформації.

Для цього він використовує додаток, який по запиту контролюючого модулю виконує операції з базою даних.

Увесь компонент модулю збереження представлений на рисунку 3.8.

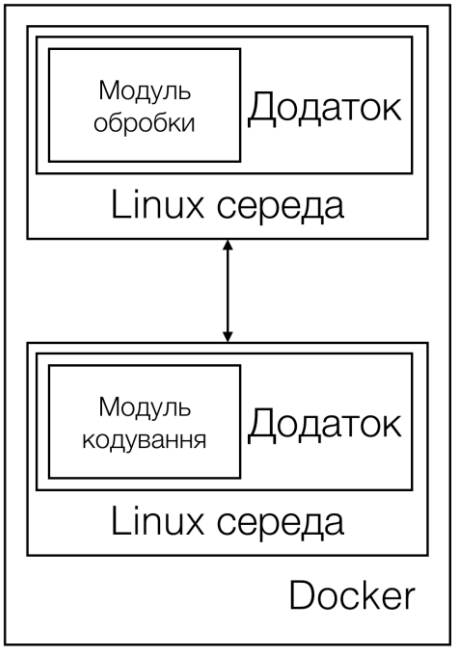

Рисунок 3.7 – Docker схема контролюючого модулю

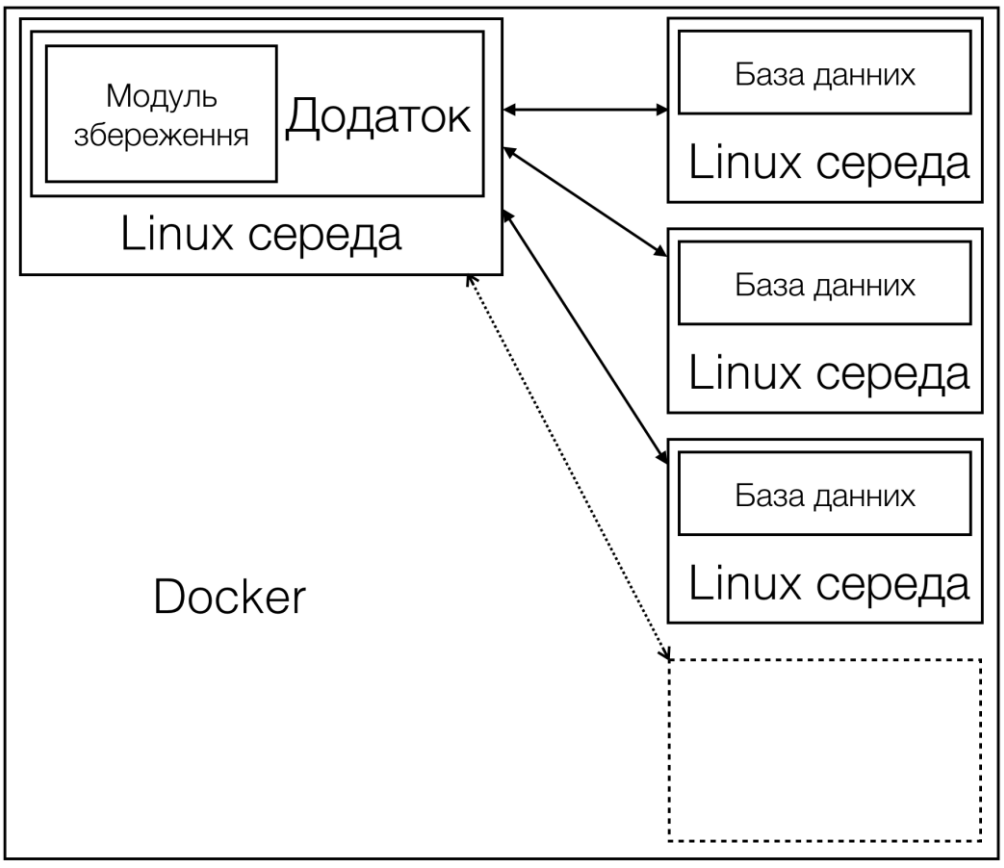

Рисунок 3.8 – Схема модулю збереження

3.3 Система обробки версій

Спочатку визначимо з яких частин складається система обробки версій . Для цього представимо схему, яка буде відображати загальну роботу цього модулю, не торкаючись усієї моделі, так як нам потрібно буде тестувати тільки ту частину, яку виконує потрібний для тестування модуль.

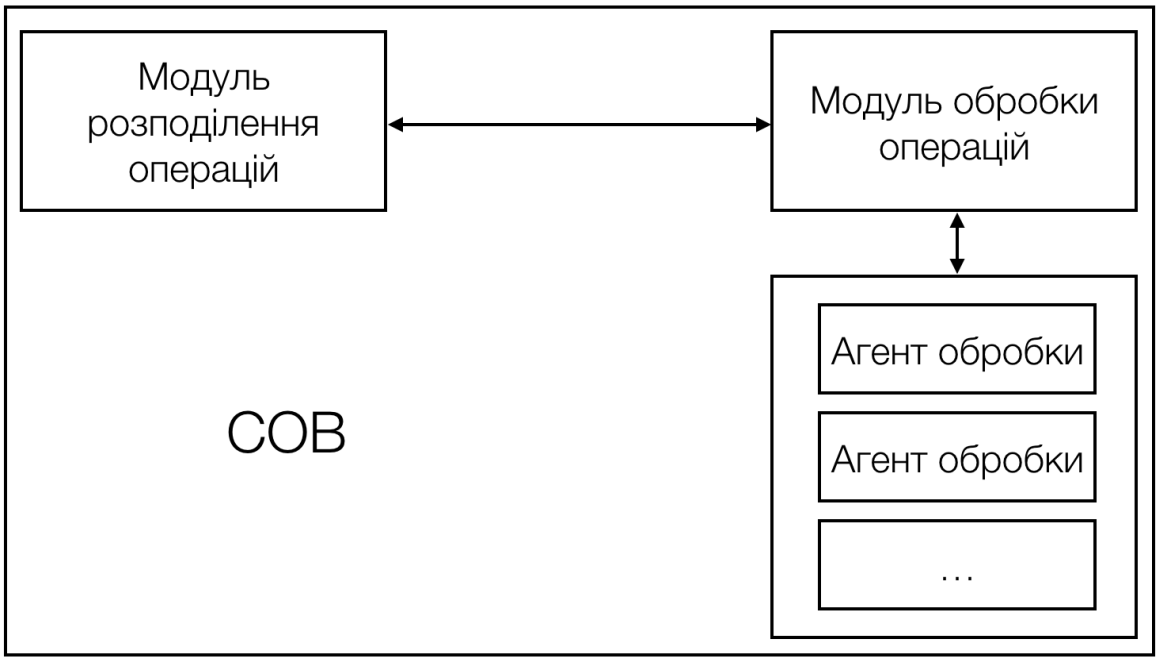

Рисунок 3.9 – Модель СОВ

Далі розглянемо детальну схему СОВ - такі компоненти, як Агент, Модуль обробки операцій. Модуль розподілення операцій.

Почнемо з того, як реалізована система обробки проектів, тобто агенти. Агент – це модуль, який відповідає лише за обробку малої неділимої операції. Це зроблено для того, щоб могли розподіляти навантаження на усю систему та розподіляти кожну роботу обробки проекту на декілька агентів. А після завершення їх роботи, використовуючи їх відповіді, формувати результат обробки.

Тобто, будемо кожну нашу роботу ділити на множину з невеликих задач та відправляти їх на обробку агентам. Схему агенту представлена на рисунку 3.10. Можемо зазначити, що агенти є компонентами, кількість яких може зростати і зменшуватись без втрат. Можно додати до системи будь-яку кількість агентів без налаштування, і вони будуть працювати за принципом, який дозволяє знаходити їх та одразу використовувати.

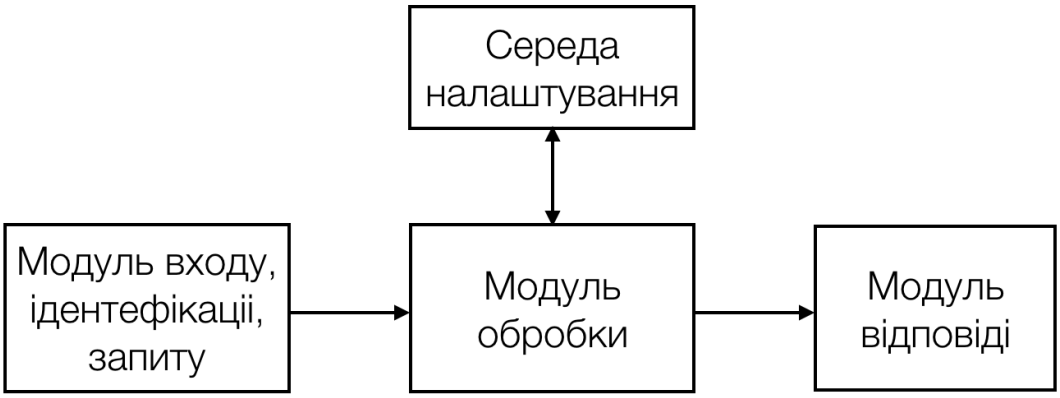

Рисунок 3.10 – Програмна схема агенту

Розглянемо цю схему більш детально.

Будемо використовувати модуль входу як точку входу (від англійської “entry point”) до агенту. Цей модуль буде приймати як запит ту частину, яку треба обробити, буде перевіряти, чи є у запитуючого права на цю операцію та, якщо все добре, цей модуль буде передавати запит далі до модулю обробки.

Будемо використовувати модуль обробки для опрацьовування запитів, які будуть надходити з модулю ідентифікації та входу. Для опрацьовування запитів цей модуль буде брати конфігурації з середи налаштування (де будуть зберігатися усі конфігураційні файли) та потім, використовуючи апаратну середу, цей модуль буде опрацьовувати кожен запит і передавати результат до модулю відповіді.

Для опрацьовування запитів цей модуль буде використовувати певні методи та алгоритми, які потрібні для цього. Ця інформація також буде приходити з середи налаштування. Та буде у собі мати компоненти обробки чи правила, за якими потрібно обробляти проект. Це дозволить бути більш гнучкими та обробляти і оновлювати базу алгоритмів, не роблячи змін у модулі обробки.

Модуль відповіді дуже схожий на модуль входу. Його метою є передати результат обробки до компоненту, який зробив запит. Для цього після того, як модуль обробки виконає роботу і направить відповідь до цього модулю, треба привести відповідь до певного формату (який буде указаний при запиті) та повернути результат обробки до ініціюючого модулю.

Увесь компонент агента представлений на рисунку 3.11. Усі процеси йдуть через модуль обробки, який налаштовується, використовуючи середу налаштування. Це

дозволяє змінювати його налаштування, не змінюючи увесь модуль, у якому знаходиться модуль.

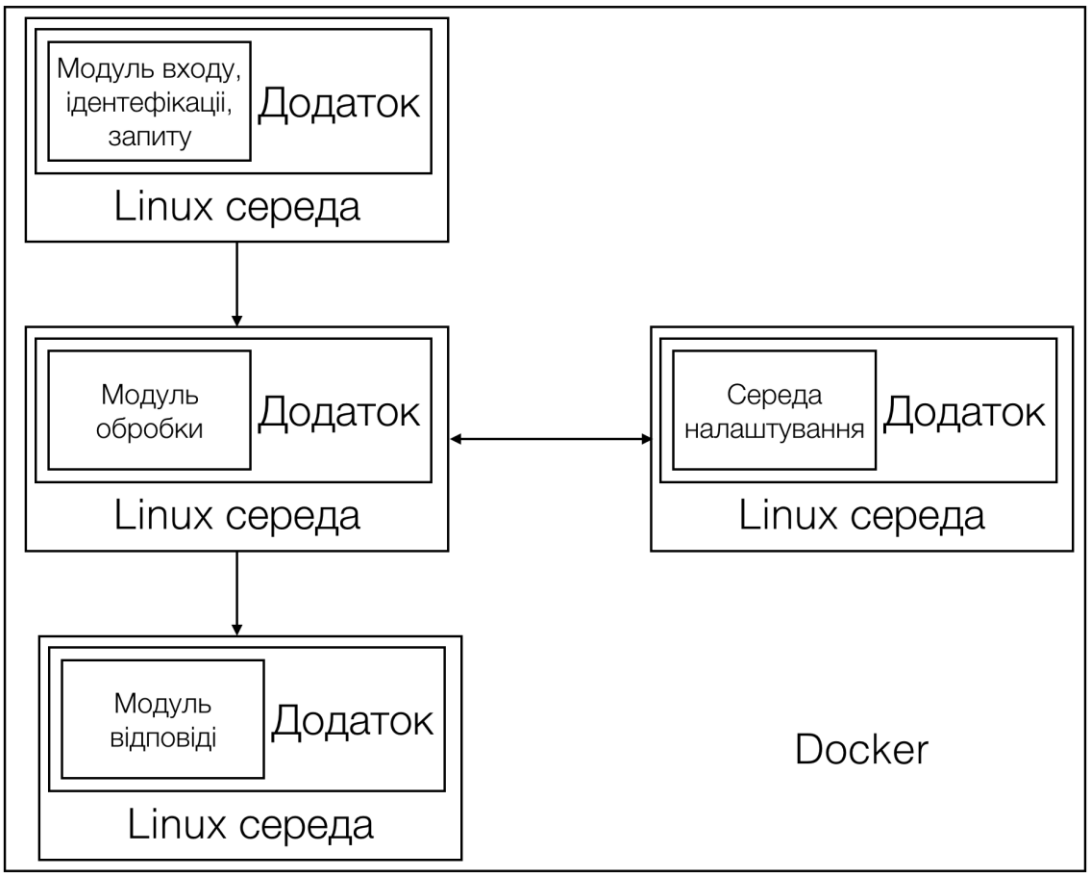

Рисунок 3.11 – Схема Docker контейнерів для агента

Модуль обробки операцій виконує роботу відносну до аналізу навантаження на агентів і розподілення роботи між ними. Він буде працювати з точки зору навантаження, а не з точки зору загальної роботи.

Робота цього модулю дуже важлива, тому що цей модуль дозволяє бути впевненими, що ресурси використовуються правильно. Це дозволяє бути більш гнучкими у випадку несподіваного зростання або спаду навантаження. В обох випадках цей модуль дозволить використовувати ресурси ефективно.

Зображення модулю обробки операцій надано на рисунку 3.12.

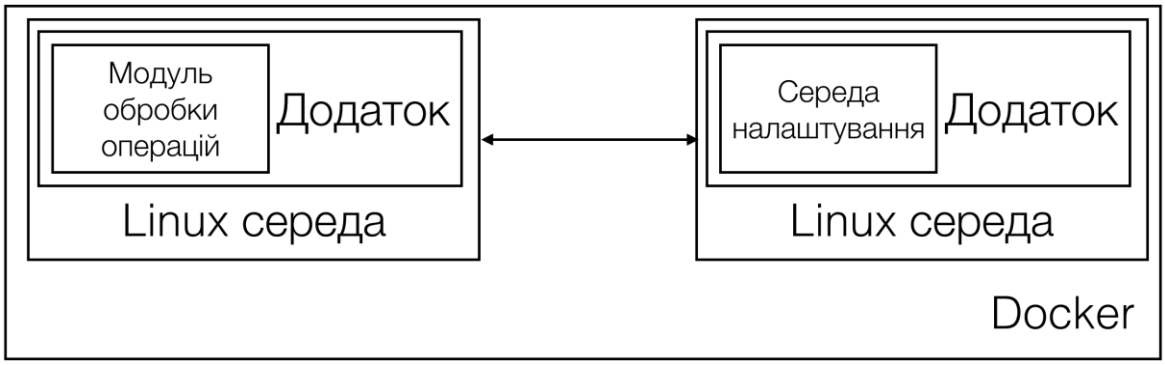

Рисунок 3.12 – Схема Docker контейнерів для модулю обробки операцій

Модуль розподілення операцій використовується СОВ для розділення кожної задачі на маленькі частини, які будуть направлені через модуль обробки операцій до агентів. Цей модуль потрібен для швидкої обробки кожної задачі. Для розподілення операцій він буде використовувати алгоритми розподілення та завдяки їм, використовуючи данні з середи налаштування, буде створювати множину дрібних задач, які можуть виконуватися паралельно.

Будемо використовувати модуль взаємодії, як компонент, який дозволяє спілкуватися у системі, а саме –поступає задача, яку необхідно виконати, для цього потрібно відіслати запити до агентів та скласти результат.

Першим кроком відправимо запит до модулю розподілення операцій, який поверне множину з дрібних задач.

Наступним кроком відправимо ці задачі до модулю обробки операцій, який у свою чергу поверне множину відповідей.

Далі цю множину відповідей направимо на модуль складання результату, який поверне результат виконання усієї роботи.

Останнім кроком потрібно повернути цей результат до модулю, який зробив запит.

Цей модуль використовується системою для обробки множини відповідей та подальшого повернення їх як комплексний результат. Для цього кожна відповідь з множини має інформацію про те, як цю інформацію об’єднати. Тобто множина поступово об’єднується та на останньому кроці увесь результат буде відправлений до модулю взаємодії.

Треба зауважити, що ця операція, так як і операція розподілення, також займають час, але перевагою є те, що обчислення будуть виконуватися паралельно, що у результаті буде набагато швидше ніж послідовна обробка. Зображення модулю розподілення операцій надано на рисунку 3.13.

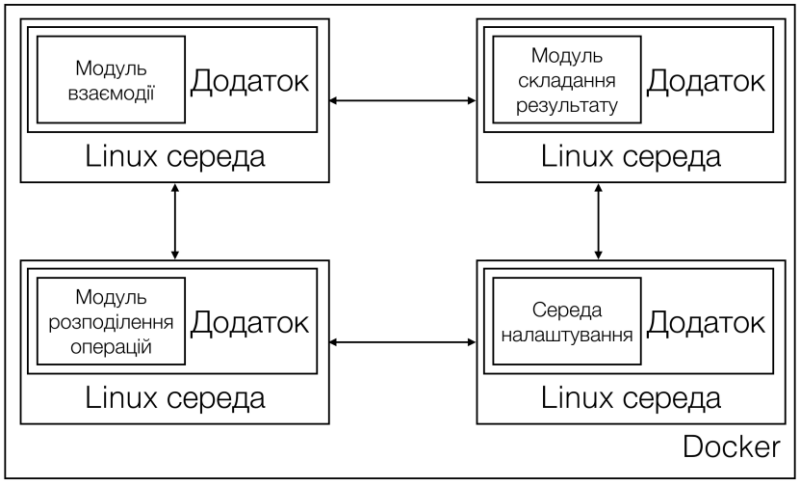

Рисунок 3.13 – Схема Docker контейнерів для модулю розподілення операцій

3.4 Порівняння схем сегментації

Загальні схеми сегментування відео, які використовуються в розподіленому кодуванні, включають в себе схему сегментування на основі вузлів (NCPS - Node Count-based Segmentation Scheme) і схему сегментації на основі GOP (GPS). Ці дві схеми відеосегментації мають кілька особливостей в процесах їх кодування (рис. 3.14).

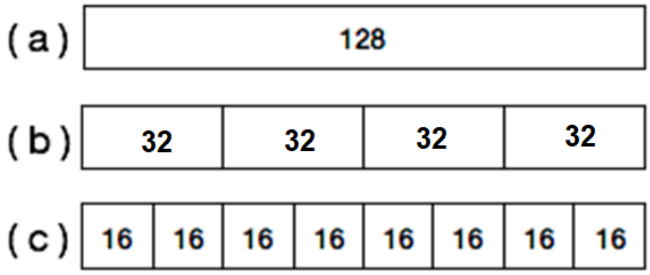

Рисунок 3.14 - Схеми сегментації: (a) Необроблене відео, (b) Node based segmentation, (c) GOP based segmentation

На рисунку 3.14 показаний простий приклад, де NCPS і GPS використовувалися для сегментації 128-кадрів відео в розподіленій системі кодування, що складається з чотирьох агентів. На рисунку показаний процес сегментації відео з використанням NCPS, який сегментує відео рівномірно на основі номера вузлів, які виконують кодування. На рисунку 3.14 (b) показано, що 128-кадрове відео було сегментовано на чотири сегменти з 32 кадрів, оскільки воно було закодовано на чотирьох вузлах. Таким чином, розподілене

кодування з використанням NCPS в кінцевому підсумку привласнює один сегмент кожному вузлу кодування, навіть коли число вузлів кодування змінюється і розмір сегмента відрізняється.

GPS - це інша основна схема сегментації. Вона сегментує відео відповідно до структури GOP, як показано на рисунку 3.14 (c). На рисунку показано, що 128-кадрове відео було в кінцевому рахунку сегментовано на 8 сегментів з 16 кадрів на основі розміру GOP 16.

На відміну від NCPS, GOP виконує процес сегментації без урахування кількості вузлів, створює таку ж кількість сегментів, що і GOP розмір і призначає їх вузлів кодування. Якщо кількість сегментів перевищує кількість вузлів кодування, кожному вузлу призначається кілька сегментів для виконання кодування.

На рисунку 3.15 представлено порівняння часу обробки відео при використанні різних алгоритмів сегментації (схем NCPS і GPS, описаних вище).

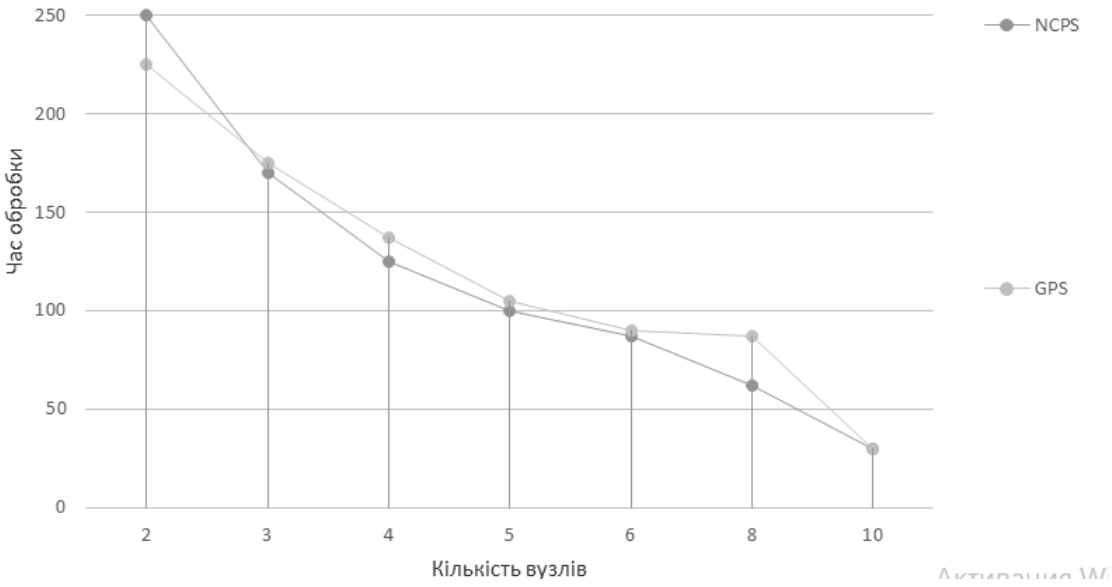

Рисунок 3.15 - Порівняння часу обробки відео при використанні різних алгоритмів сегментації

3.5 Тестування

Далі проведемо тестування частини реалізованої системи.

У першу чергу треба провести тестування клієнту і виконати, використовуючи його, декілька операцій, які необхідні у кожній системі керування версіями.

Першим кроком у будь-якої системі керування версіями є ініціалізація проекту для подальшого його використовування. Проект потрібно ініціалізувати лише один раз на самому початку, а потім усі конфігурації та інша інформація будуть доступні постійно. Приклад ініціалізації наведено на рисунку 3.16.

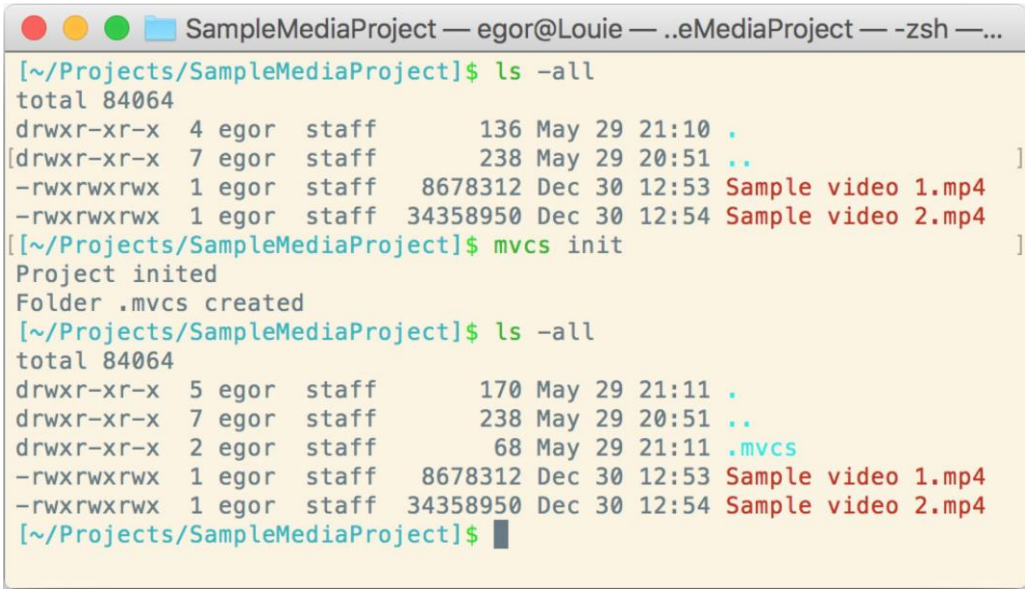

Рисунок 3.16 – Скріншот процесу ініціалізації

Наступний крок – додати файли до індексу, який система буде використовувати для операцій над проектом. Операції додавання файлів до індексу, а також перевірка роботи функції статусу представлені на рисунку 3.17. Як можно бачити, ця операція доволі проста і прозора.

Рисунок 3.17 – Скріншот процесу додавання файлів до індексу

Також реалізовані такі допоміжні функції, які спрощують роботу з системою. Наприклад функція, яка дозволяє перевірити версію клієнта, встановлену у споживача нашого продукту. Ще система має функцію перевірки авторів (для зворотного зв'язку). Функція допомоги, яка містить список команд, також доступна з клієнту. Перевірка цих функцій надана на рисунку 3.18.

Рисунок 3.18 – Скріншот допоміжних функцій системи

Далі можно перевірити сервер.

Сервер працює у контейнері Jetty, та написаний на мові програмування Java, використовуючи фреймворк Spring. Також було застосовано бібліотеку swagger, яка дозволяє переглядати методи серверу у зручному форматі.

Спочатку надамо схему контролера, який використовується зараз нашою системою на рисунку 3.19.

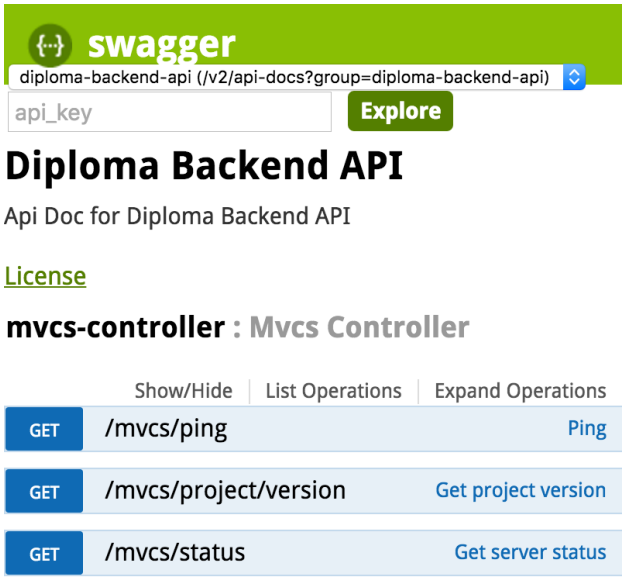

Рисунок 3.19 – Загальна схема контролеру сервера

Далі зазначимо кожен з методів окремо.

Ping – це достатньо розповсюджений метод на серверах, так як він лише каже чи працює сервер, чи ні, не виконуючи складних операцій. Тому можемо запитувати його стільки, скільки цього бажаємо. Однак цей метод є важливим, так як можемо будувати графіки, за допомогою яких перевіряти, що сервер працює, і є до нього доступ. Якщо ping не поверне нам відповідь, то це зразу буде означати, що сервер у цей час недоступний, і потрібно це терміново виправляти. Детальна схема відповіді з методу Ping надана на рисунку 3.20.

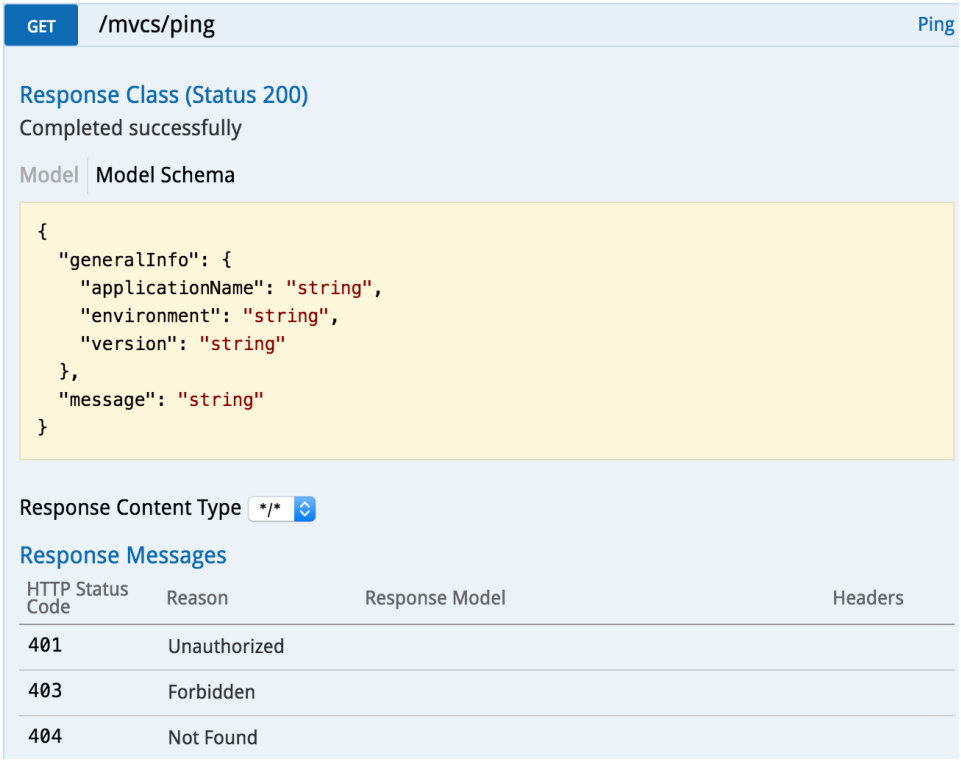

Рисунок 3.20 – Схема відповіді методу ping

Також маємо можливість перевірити результат методу ping. Результат надано на рисунку 3.22.

Як бачимо, сервер повернув не лише відповідь та й загальну інформацію щодо сервера, таку як – назва, версія, обладнання. Ця частина є у будь-якій відповіді і потрібна для того, щоб у разі дефекту завжди мати можливість перевірити де саме і чим він був викликаний.

Status – це розвернута версія методу ping. Вона значно складніша та надає допоміжну інформацію, не лише робить сервер чи ні. Завдяки цьому методу можно дізнатись також інформацію про проміжні модулі, до яких цей сервер має доступ та які використовує.

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:9999/mvcs/ping'
```

Request URL

```
http://localhost:9999/mvcs/ping
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body

```
{
  "generalInfo": {
    "applicationName": "diploma-backend",
    "version": "0.0.1a",
    "environment": "dev"
  },
  "message": "Pong"
}
```

Response Code

```
200
```

Response Headers

```
{
  "date": "Wed, 01 Jun 2016 02:51:46 GMT",
  "server": "Jetty(9.2.15.v20160210)",
  "transfer-encoding": "chunked",
  "content-type": "application/json;charset=UTF-8"
}
```

Рисунок 3.22 – Приклад відповіді сервера на метод ping

Також завдяки цієй функції, при використанні таких бібліотек як hystix та javamelody маємо можливість надати користувачу ще детальнішу інформацію щодо процесів серверу у цей час, таких як навантаження, кількість операцій, потоків, запитів, відповідей, тощо. Схема відповіді у методу status така як і у ping. У реалізованій частині системи цей метод зараз повертає фактичний статус системи.

Приклад надано на рисунку 3.23.

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:9999/mvcs/status'
```

Request URL

```
http://localhost:9999/mvcs/status
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body

```
{
  "generalInfo": {
    "applicationName": "diploma-backend",
    "version": "0.0.1a",
    "environment": "dev"
  },
  "message": "Running"
}
```

Response Code

```
200
```

Response Headers

```
{
  "date": "Wed, 01 Jun 2016 03:11:07 GMT",
  "server": "Jetty(9.2.15.v20160210)",
  "transfer-encoding": "chunked",
  "content-type": "application/json;charset=UTF-8"
}
```

Рисунок 3.23 – Приклад відповіді сервера на метод status

Останнім з цих методів є метод project version, який складніше ніж попередні два, та також має трохи інакшу структуру. Цей метод використовується зараз як приклад того, як можливо перевіряти, яка версія проекту знаходиться на сервері.

Схема цього методу надана на рисунку 3.24.

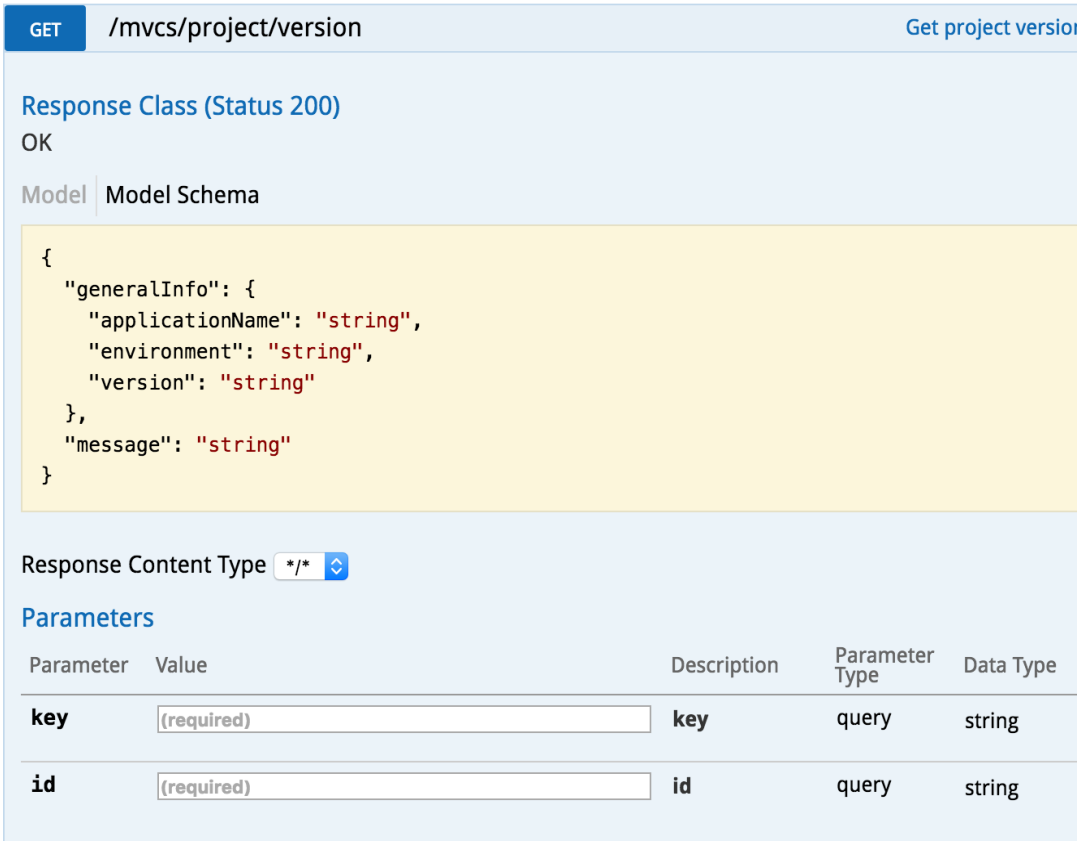

Рисунок 3.24 – Схема методу project version

Як бачимо, схема цього методу має спільні частини з іншими, тому що він також повертає лише одне повідомлення. Але треба зауважити, що наш метод зараз має 2 нових параметри. Перший, key - ключ, використовується для ідентифікації користувача, для того, щоб персональна інформація не була доступна іншим окрім власників проекту. Другий параметр – це унікальний ідентифікатор проекту, версію якого бажаємо переглянути. Приклад успішного запиту надано на рисунку 3.25.

У разі, якщо ми не авторизовані, запит не буде оброблятися, це також можно перевірити. Результат надано на рисунку 3.26.

Request URL

```
http://localhost:9999/mvcs/project/version?key=52a1f0243004a10eabc8da12&id=5357d5fee4b0ebfcbf845781
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body

```
{
  "generalInfo": {
    "applicationName": "diploma-backend",
    "version": "0.0.1a",
    "environment": "dev"
  },
  "message": "12"
}
```

Response Code

```
200
```

Рисунок 3.25 – Приклад авторизованої відповіді з сервера

Request URL

```
http://localhost:9999/mvcs/project/version?key=fake_key&id=5357d5fee4b0ebfcbf845781
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body

```
no content
```

Response Code

```
403
```

Рисунок 3.26 – Приклад неавторизованої відповіді з сервера

Порівняємо час обробки відео, базуючись на кількісті агентів (рис 3.27), та побудуємо графік залежності (рис 3.28).

```
Downloads — -bash — 80×33
su-macbook-57fb:Downloads emasalitin$ ./encoder_diploma 1 test_video.mp4
Encoding 'test_video.mp4' node count - 1:
.............................................
Success (3907s)
su-macbook-57fb:Downloads emasalitin$ ./encoder_diploma 2 test_video.mp4
Encoding 'test_video.mp4' node count - 2:
.............................................
Success (2012s)
su-macbook-57fb:Downloads emasalitin$ ./encoder_diploma 1 test_video.mp4
Encoding 'test_video.mp4' node count - 1:
.............................................
Success (3907s)
su-macbook-57fb:Downloads emasalitin$ ./encoder_diploma 2 test_video.mp4
Encoding 'test_video.mp4' node count - 2:
.......................
Success (2012s)
su-macbook-57fb:Downloads emasalitin$ ./encoder_diploma 4 test_video.mp4
Encoding 'test_video.mp4' node count - 4:
.............
Success (1120s)
su-macbook-57fb:Downloads emasalitin$ ./encoder_diploma 8 test_video.mp4
Encoding 'test_video.mp4' node count - 8:
.......
Success (502s)
su-macbook-57fb:Downloads emasalitin$ ./encoder_diploma 12 test_video.mp4
Encoding 'test_video.mp4' node count - 12:
.....
Success (330s)
su-macbook-57fb:Downloads emasalitin$ ./encoder_diploma 20 test_video.mp4
Encoding 'test_video.mp4' node count - 20:
..
Success (190s)
su-macbook-57fb:Downloads emasalitin$
```

Рисунок 3.27 – Обчислення часу обробки відео

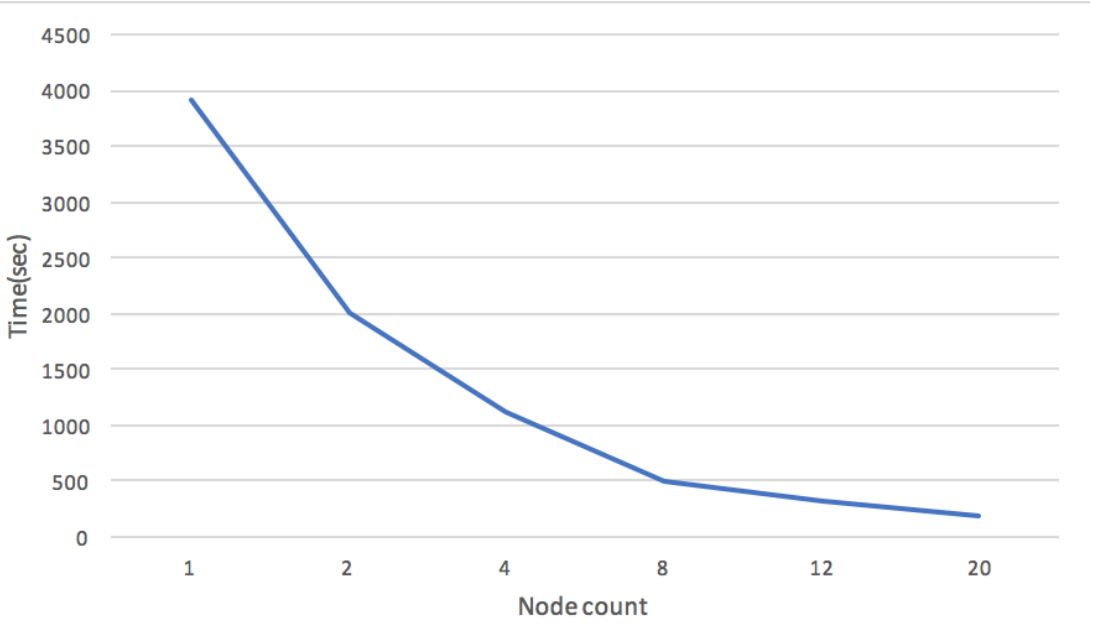

Рисунок 3.28 – Графік залежності кількісті нод від часу виконання

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

В даному розділі проведено аналіз потенційних небезпечних та шкідливих виробничих факторів. Розглянуті умови, що дозволяють забезпечити гігієну праці, виробничу санітарію. Розроблено заходи з техніки безпеки та рекомендації з пожежної профілактики. Аналіз потенційно небезпечних і шкідливих виробничих чинників виконується для робочих умов, з використанням персонального комп'ютера, на якому буде йти розробка.

4.1 Загальні питання з охорони праці

Умови праці на робочому місці, безпека технологічних процесів, машин, устаткування та інших засобів виробництва, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці. В законі України «Про охорону праці» визначається, що охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

4.2 Аналіз стану умов праці

Робота над проектом проходитиме в приміщенні багатоквартирного будинку. Для даної роботи достатньо однієї людини, для якої надано робоче місце зі стаціонарним комп’ютером.

4.2.1 Вимоги до приміщень

Геометричні розміри приміщення зазначені в табл. 4.1.

Таблиця 4.1 – Розміри приміщення

Найменування	**Значення**
Довжина, м	5
Ширина, м	5
Висота, м	3
Площа, $м^2$	25
Об’єм, $м^3$	75

Згідно з [30] розмір площі для одного робочого місця оператора персонального комп’ютера має бути не менше 6 кв. м, а об’єм — не менше 20 куб. м. Отже, дане приміщення цілком відповідає зазначеним нормам.

4.2.2 Вимоги до організації місця праці

При порівнянні відповідності характеристик робочого місця нормативним основні вимоги до організації робочого місця за [28] (табл. 4.2) і відповідними фактичними значеннями для робочого місця, констатуємо повну відповідність.

Таблиця 4.2 - Характеристики робочого місця

Найменування параметра	Фактичне значення	Нормативне значення
Висота робочої поверхні, мм	750	680 ÷800
Висота простору для ніг, мм	730	не менше 600
Ширина простору для ніг, мм	660	не менше 500
Глибина простору для ніг, мм	700	не менше 650
Висота поверхні сидіння, мм	470	400 ÷500
Ширина сидіння, мм	400	не менше 400
Глибина сидіння, мм	400	не менше 400
Висота поверхні спинки, мм	600	не менше 300
Ширина опорної поверхні спинки, мм	500	не менше 380
Радіус кривини спинки в горизонтальній площині, мм	400	400
Відстань від очей до екрану дисплея, мм	800	700 ÷800

Температура в приміщенні протягом року коливається у межах 18–24°С, відносна вологість — близько 50%. Швидкість руху повітря не перевищує 0,2 м/с. Шум в лабораторії знаходиться на рівні 50 дБА. Система вентилювання приміщення — природна неорганізована, а опалення — централізоване.

4.2.3 Навантаження та напруженість процесу праці

За фізичним навантаженням робота відноситься до категорії легкі роботи (Іа), її виконують сидячи з періодичним ходінням. Щодо характеру організування виконання дипломної роботи, то він підпадає під нав’язаний режим, оскільки певні розділи роботи необхідно виконати у встановлені конкретні терміни.

Роботу за дипломним проектом визнано, такою, що займає 50% часу робочого дня та за восьмигодинної робочої зміни рекомендовано встановити додаткові регламентовані перерви - для розробників програм тривалістю 15 хв. через кожну годину роботи;

4.3 Виробнича санітарія

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки можуть бути надалі вирішені питання необхідності забезпечення працюючих достатньою кількістю освітлення, вентиляції повітря, організації заземлення, тощо.

4.3.1 Аналіз небезпечних та шкідливих факторів при виробництві (експлуатації) виробу

Аналіз небезпечних та шкідливих виробничих факторів виконується у табличній формі (табл. 5.3). Роботу, пов'язану з ЕОП з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ЕОМ з ВДТ і ПП, виконують із забезпеченням виконання НПАОП 0.00.-1.28-10 [36], які встановлюють вимоги безпеки до обладнання робочих місць, до роботи із

застосуванням ЕОМ з ВДТ і ПП. Основними робочими характеристиками персонального комп'ютера є:

- робоча напруга U=+220В +-5%;
- робочий струм I=2А;
- споживана потужність Р=350 Вт.

Таблиця 4.3 – Аналіз небезпечних і шкідливих виробничих факторів

Небезпечні і шкідливі виробничі фактори	Джерела факторів (види робіт)	Кількісна оцінка	Нормативні документи
1	2	3	4
Фізичні			
- підвищена температура поверхонь обладнання	експлуатація ЕОМ, принтерів, сканерів чи/або серверного обладнання для роботи	2	[30]
- підвищений рівень шуму на робочому місці	-//-	2	[29]
- підвищений рівень вібрації	-//-	2	[30] [43]
- підвищена або знижена вологість повітря	-//-	2	[30]
- підвищена або знижена рухливість повітря	-//-	1	[30]
- підвищений рівень іонізуючого випромінення в робочій зоні	-//-	2	[30] [40]
- підвищений рівень електромагнітного випромінення	-//-	2	[40]
- підвищений рівень напруги електричної мережі, замикання якої може відбутися через тіло людини	-//-	4	[41] [30]
- підвищений рівень статичної електрики	-//-	2	[41]
- підвищена напруженість електричного поля	-//-	2	[40]
- підвищена напруженість магнітного поля	-//-	2	[40]
- недостатність	порушення умов праці (вимог до	2	[27]

природного світла	приміщень)		
- недостатнє освітлення робочої зони	порушення гігієнічних параметрів виробничого середовища	3	[27]
- підвищена яскравість світла	порушення умов праці (організації місця праці-налагодження моніторів)	1	[28]
Продовження таблиці 4.3 - понижена контрастність	-//-	1	[28]
психофізіологічні:			
- нервово-психічна перевантаження (розумове, перенапруження аналізаторів-зорових)	- пошук інформації для постановки теми; - пошук та аналіз аналогів і літератури; - пошук наявних технологій, моделювання та аналіз алгоритмів; - виконання роботи за темою диплома, тестування; - оформлення роботи	4	[42] [28]
- фізичні (статичне сидіння)	порушення умов праці (організації місця праці - сидіння користувача) та організації робочого часу - безпервна робота)	2	[42] [28]

Робочі місця мають відповідати вимогам Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджених постановою Головного державного санітарного лікаря України від 10.12.98 N 7 [28].

4.3.2 Пожежна безпека

Небезпека розвитку пожежі на обчислювальному центрі обумовлюється застосуванням розгалужених систем електроживлення ЕОМ, вентиляції і кондиціювання.

Запобігти утворенню горючого середовища (замінити горючі речовини і матеріали на негорючі і важкогорючі) не надається технічно можливим. Тому проектом передбачаються способи і засоби запобігання утворення (або внесення) в горюче середовище джерел запалювання, таких як:

1) застосування електроустаткування, відповідної пожежонебезпечної і вибухонебезпечної зонам відповідно до ПУЕ;

2) застосування в конструкції швидкодійних засобів захисного відключення можливих джерел запалення;

3) виключення можливості появи іскрового розряду в горючому середовищі з енергією, рівної і вище мінімальної енергії запалення.

Згідно [34] таке приміщення, площею 25 м^2, відноситься до категорії "В" (пожежонебезпечної) та для протипожежного захисту в ньому проектом передбачено устаткування автоматичною пожежною сигналізацією із застосуванням датчиків-сповіщувачів РІД-1 (сповіщувач димовий ізоляційний) в кількості 1 шт., і застосуванням первинних засобів пожежогасіння.

Простори усередині приміщень в межах, яких можуть утворюватися або знаходиться пожежонебезпечні речовини і матеріали відповідно до [34] відносяться до пожежонебезпечної зони класу П-IIа. Це обумовлено тим, що в приміщенні знаходяться тверді горючі та важкозаймисті речовини та матеріали. Приміщенню, у якому розташоване робоче місце, присвоюється II ступень вогнестійкості.

4.3.3 Електробезпека

На робочому місці виконуються наступні вимоги електробезпеки: ПК, периферійні пристрої та устаткування для обслуговування, електропроводи і кабелі за виконанням та ступенем захисту відповідають класу зони за ПУЕ (правила улаштування електроустановок), мають апаратуру захисту від струму короткого замикання та інших аварійних режимів. Лінія електромережі для живлення ПК, периферійних пристроїв і устаткування для обслуговування, виконана як окрема групова три- провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Штепсельні з'єднання та електророзетки крім контактів фазового та нульового робочого провідників мають спеціальні контакти для підключення нульового захисного провідника. Електромережа штепсельних розеток для живлення персональних ПК укладено по підлозі поруч зі стінами відповідно до затвердженого плану розміщення обладнання та технічних характеристик обладнання. Металеві труби та гнучкі металеві

рукави заземлені. Захисне заземлення включає в себе заземлюючих пристроїв і провідник, який з'єднує заземлюючий пристрій з обладнанням, яке заземлюється - заземлюючий провідник.

4.4 Гігієнічні вимоги до параметрів виробничого середовища

4.4.1 Мікроклімат

Мікроклімат робочих приміщень – це клімат внутрішнього середовища цих приміщень, що визначається діючої на організм людини з'єднанням температури, вологості, швидкості переміщення повітря.

Дане приміщення обладнане системами опалення, кондиціонування повітря або припливно-витяжною вентиляцією. У приміщенні на робочому місці забезпечуються оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря у відповідності до [29]. Рівні позитивних і негативних іонів у повітрі мають відповідати [29].

Контроль параметрів мікроклімату в холодний і теплий період року здійснюється не менше 3-х разів на зміну (на початку, середині, в кінці).

4.4.2 Освітлення

Світло є природною умовою існування людини. Воно впливає на стан вищих психічних функцій і фізіологічні процеси в організмі. Хороше освітлення діє тонізуюче, створює гарний настрій, покращує протікання основних процесів вищої нервової діяльності.

У приміщенні, де розташовані ЕОМ передбачається природне бічне освітлення[27]. Джерелом природного освітлення є сонячне світло. Регулярно повинен проводитися контроль освітленості, який підтверджує, що рівень освітленості задовольняє ДБН і для даного приміщення в світлий час доби достатньо природного освітлення.

Розрахунок освітлення.

Для виробничих та адміністративних приміщень світловий коефіцієнт приймається не менше -1/8, в побутових – 1/10:

$$S_b = \left(\frac{1}{5} \div \frac{1}{10}\right) \cdot S_n, \quad (4.1)$$

де S_b – площа віконних прорізів, м^2;

S_n – площа підлоги, м^2.

$S_n = a \cdot b = 5 \cdot 5 = 25$ м^2,

$S = 1/8 \cdot 25 = 3{,}125$ м^2.

Приймаємо 2 вікна площею S=1,6 м^2 кожне.

Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні.

Розрахунок кількості світильників n виробляється по формулі (4.2):

$$n = \frac{E \cdot S \cdot Z \cdot K}{F \cdot U \cdot M}, \quad (4.2)$$

де E – нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S – освітлювана площа, м^2; S = 25 м^2;

Z – поправочний коефіцієнт світильника (Z = 1,15 для ламп розжарювання та ДРЛ; Z = 1,1 для люмінесцентних ламп) приймаємо рівним 1,1;

K – коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації – 1,5;

U – коефіцієнт використання, залежний від типу світильника, показника індексу приміщення і т.п. – 0,575

M – число люмінесцентних ламп в світильнику – 2;

F – світловий потік лампи – 5400лм (для ЛБ-80).

Підставивши числові значення у формулу (4.2), отримуємо:

$$n = \frac{300 \cdot 25 \cdot 1{,}1 \cdot 1{,}5}{5400 \cdot 0{,}575 \cdot 2} \approx 2{,}0$$

Приймаємо освітлювальну установку, яка складається з 2-х світильників, які складаються з двох люмінесцентних ламп загальною потужністю 160 Вт, напругою – 220 В.

4.4.3 Шум та вібрація, електромагнітне випромінювання

Рівень шуму, що супроводжує роботу користувачів персональних комп'ютерів, коливається у межах 50–65 дБА [29]. У залах комп'ютерного набору рівні шуму не повинні перевищувати 65 дБА.

Віброізоляція можливо здійснювати за допомогою спеціальної прокладки під системний блок, який послаблює передачу вібрацій робочому столу. Вібрація на робочому місці, що розглядається, відповідає нормам [29].

4.4.4 Вентилювання

У приміщенні, де знаходяться ЕОМ, повітрообмін реалізується за допомогою природної організованої вентиляції. Цей метод забезпечує приток потрібної кількості свіжого повітря, що визначається в СНіП.

Також має здійснюватися провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.5 Заходи з організації виробничого середовища та попередження виникнення надзвичайних ситуацій

Відповідно до санітарно-гігієнічних нормативів та правил експлуатації обладнання наводимо приклади деяких заходів безпеки.

Заходи безпеки під час експлуатації персонального комп'ютера та периферійних пристроїв передбачають:

- правильне організування місця праці та дотримання оптимальних режимів праці та відпочинку під час роботи з ПК;
- експлуатацію сертифікованого обладнання;
- дотримання заходів електробезпеки;

- забезпечення оптимальних параметрів мікроклімату;
- забезпечення раціонального освітлення місця праці;
- облаштовуючи приміщення для роботи з ПК, потрібно передбачити припливно-витяжну вентиляцію або кондиціювання повітря.

Вимоги безпеки при надзвичайних ситуаціях:

1) При раптовому припиненні подачі електричної енергії вимкнути ПК. Витягнути вилки з розеток. При наявності ознак горіння (дим, запах горілого) необхідно вимкнути ПК, знайти місце загоряння і виконати всі можливі заходи для його ліквідації.

2) При замиканні, перевантаженні електричного струму на електричному обладнанні, внаслідок ураження грозової блискавки та ймовірної небезпеки ураженням електричним струмом, приймають наступне:

- попередження замикання здійснюється правильним вибором, монтажем експлуатації мереж;
- застосування захисту схем у вигляді швидкодіючих реле, а також вимикачів, плавких запобіжників.

Розрахунок захисного заземлення (забезпечення електробезпеки будівлі).

Згідно з класифікацією приміщень за ступенем небезпеки ураження електричним струмом [35], приміщення в якому проводяться всі роботи відноситься до першого класу (без підвищеної небезпеки). Коефіцієнт використання вертикальних заземлювачів $\eta_в$ в залежності від розміщення заземлювачів та їх кількості знаходиться в межах 0,4…0,99. Взаємну екрануючу дію горизонтального заземлювача (з'єднувальної смуги) враховують за допомогою коефіцієнта використання горизонтального заземлювача η_c.

Послідовність розрахунку.

1) Визначається необхідний опір штучних заземлювачів $R_{шт.з.}$:

$$R_{шт.з.} = \frac{R_д \cdot R_{пр.з.}}{R_{пр.з.} - R_д}, \qquad (4.3)$$

де $R_{пр.з.}$ – опір природних заземлювачів; $R_д$ – допустимий опір заземлення. Якщо природні заземлювачі відсутні, то $R_{шт.з.}=R_д$.

Підставивши числові значення у формулу (4.3), отримуємо:

$$R_{шт.з.} = \frac{4 \cdot 40}{40 - 4} \approx 4 \text{ Ом}$$

2) Опір заземлення в значній мірі залежить від питомого опору ґрунту ρ, Ом·м. Приблизне значення питомого опору глини приймаємо ρ=40 Ом•м (табличне значення).

3) Розрахунковий питомий опір ґрунту, $\rho_{розр}$, Ом·м, визначається відповідно для вертикальних заземлювачів $\rho_{розр.в}$, і горизонтальних $\rho_{розр.г}$, Ом·м за формулою:

$$\rho_{розр..} = \psi \cdot \rho, \tag{4.4}$$

де ψ – коефіцієнт сезонності для вертикальних заземлювачів I кліматичної зони з нормальною вологістю землі, приймається для вертикальних заземлювачів $\rho_{розр.в}$=1,7 і горизонтальних $\rho_{розр.г}$=5,5 Ом·м.

$$\rho_{розр.в} = 1,7 \cdot 40 = 68 \text{ Ом•м}$$

$$\rho_{розр.г} = 5,5 \cdot 40 = 220 \text{ Ом•м}$$

4) Розраховується опір розтікання струму вертикального заземлювача $R_в$, Ом, за (4.5).

$$R_в = \frac{\rho_{розр.в}}{2 \cdot \pi \cdot l_в} \cdot \left(\ln \frac{2 \cdot l_в}{d_{ст}} + \frac{1}{2} \cdot \ln \frac{4 \cdot t + l_в}{4 \cdot t - l_в} \right), \tag{4.5}$$

де $l_в$ – довжина вертикального заземлювача (для труб - 2−3 м; $l_в$=3 м);

$d_{ст}$ – діаметр стержня (для труб - 0,03−0,05 м; $d_{ст}$=0,05 м);

t – відстань від поверхні землі до середини заземлювача, яка визначається за ф. (4.6):

$$t = h_в + \frac{l_в}{2}, \tag{4.6}$$

де $h_в$ – глибина закладання вертикальних заземлювачів (0,8 м); тоді $t = 0,8 + \frac{3}{2} = 2,3$ м

$$R_в = \frac{68}{2 \cdot \pi \cdot 3} \cdot \left(\ln \frac{2 \cdot 3}{0,05} + \frac{1}{2} \cdot \ln \frac{4 \cdot 2,3 + 3}{4 \cdot 2,3 - 3} \right) = 18,5 \text{ Ом}$$

5) Визначається теоретична кількість вертикальних заземлювачів n штук, без урахування коефіцієнта використання $\eta_в$:

$$n = \frac{2 \cdot R_в}{R_д} = \frac{2 \cdot 18,5}{4} = 9,25 \quad (4.7)$$

І визначається коефіцієнт використання вертикальних електродів групового заземлювача без врахування впливу з'єднувальної стрічки $\eta_в$ =0,57 (табличне значення).

6) Визначається необхідна кількість вертикальних заземлювачів з урахуванням коефіцієнта використання $n_в$, шт:

$$n_в = \frac{2 \cdot R_в}{R_д \cdot \eta_в} = \frac{2 \cdot 18,5}{4 \cdot 0,57} = 16,2 \approx 16 \quad (4.8)$$

7) Визначається довжина з'єднувальної стрічки горизонтального заземлювача $l_с$, м:

$$l_с = 1,05 \cdot L_в \cdot (n_в - 1), \quad (4.9)$$

де $L_в$ – відстань між вертикальними заземлювачами, (прийняти за $L_в$ = 3м);

$n_в$ – необхідна кількість вертикальних заземлювачів.

$$l_с = 1,05 \cdot 3 \cdot (16 - 1) \approx 48 \text{м}$$

8) Визначається опір розтіканню струму горизонтального заземлювача (з'єднувальної стрічки) $R_г$, Ом:

$$R_г = \frac{\rho_{розр.г}}{2 \cdot \pi \cdot l_с} \cdot \ln \frac{2 \cdot l_с^2}{d_{см} \cdot h_г}, \quad (4.10)$$

де $d_{см}$ – еквівалентний діаметр смуги шириною b, $d_{см}$ =0,95b, b = 0,15 м;

$h_г$ – глибина закладання горизантальних заземлювачів (0,5 м);

$l_с$ - довжина з'єднувальної стрічки горизонтального заземлювача $l_с$, м

$$R_г = \frac{220}{2 \cdot \pi \cdot 48} \cdot \ln \frac{2 \cdot 48^2}{0,95 \cdot 0,15 \cdot 0,5} = 8,1 \text{ Ом}$$

9) Визначається коефіцієнт використання горизонтального заземлювача $\eta_с$. відповідно до необхідної кількості вертикальних заземлювачів $n_в$. Коефіцієнт використання з'єднувальної смуги $\eta_с$=0,3 (табличне значення).

10) Розраховується результуючий опір заземлювального електроду з урахуванням з'єднувальної смуги:

$$R_{заг} = \frac{R_в \cdot R_г}{R_в \cdot \eta_с + R_г \cdot n_в \cdot \eta_в} \le R_д. \quad (4.11)$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: $R_{заг}$ <4 Ом, а саме:

$$R_{заг} = \frac{18,5 \cdot 8,1}{18,5 \cdot 0,3 + 8,1 \cdot 16 \cdot 0,57} = 1,9 \le R_д$$

3) При виникненню пожеж при роботі на ПЕОМ від таких можливими джерел запалювання як:

- іскри і дуги коротких замикань;
- перегрів провідників, резисторів та інших радіодеталей ПЕОМ, від тривалої перевантаження та наявність перехідного опору;
- іскри при розмиканні і розмиканні ланцюгів;
- розряди статичної електрики;
- необережному поводженню з вогнем, а також вибухи газо-повітряних і паро-повітряних сумішей.

4.6 Охорона навколишнього природного середовища

4.6.1 Загальні дані з охорони навколишнього природного середовища

Діяльність за темою магістерської роботи, процес виконання якої впливає на навколишнє природнє середовище, регламентується нормами діючого законодавства: Законом України «Про охорону навколишнього природного середовища», Законом України «Про забезпечення санітарного та епідемічного благополуччя населення», Законом України «Про відходи», Законом України «Про охорону атмосферного повітря», Законом України «Про захист населення і територій від надзвичайних ситуацій техногенного та природного характеру», Водний кодекс України.

Основним екологічним аспектом в процесі діяльності за даними спеціальностями є процеси впливу на атмосферне повітря та процеси поводження з відходами, які

утворюються, збираються, розміщуються, передаються на знешкодження, утилізацію, тощо в ІТ галузі.

Вплив на атмосферне повітря при нормальних умовах праці відсутній, бо немає в приміщенні сканерів, принтерів та інших джерел викиду забруднюючих речовин в повітря робочої зони.

4.6.2 Вимоги до збору, пакування та розміщення відходів ІТ галузі

Наводяться вимоги зберігання виявлених за своєю роботою відходів відповідно до вимог Державних санітарних правил і норм ДСанПіН 2.2.7.029.

Відходи в міру їх накопичення збирають у тару, відповідну класу небезпеки, з дотриманням правил безпеки, після чого доставляють до місця тимчасового зберігання відходів відповідно до затвердженої схеми їх розміщення. Зазначені для зберігання відходів місця чи об'єкти повинні використовуватися лише для заявлених відходів.

Не допускається зберігання відходів у невстановлених схемою місцях, а також перевищення норм тимчасового зберігання відходів.

Не допускається змішування відходів різних видів і класів небезпеки з будівельними і побутовими відходами, відходами дерев'яної, металевої, синтетичної тари, відходами текстильних матеріалів (старий спецодяг, ганчірки) і ін.

Проведення заготовки, здачі, переробки та реалізації металобрухту встановлені окремо Законом України «Про металобрухт».

Особливий контроль наділяється збору і зберіганню відпрацьованих енергоощадних ламп як відходам I класу небезпеки, що збираються і обов’язково передаються на утилізацію підприємствам, що мають ліцензію на поводження з такими небезпечними відходами.

Всі відходи, що утворюються в процесі діяльності/роботи, підлягають обліку.

Під час роботи з відходами (прибирання виробничих приміщень, збір і сортування, навантаження, транспортування, розвантаження та ін.) працівники та обслуговуючий персонал підприємства повинні бути забезпечені засобами індивідуального захисту та дотримуватися вимог інструкцій з охорони праці, що діють на підприємстві.

Побутові та будівельні відходи вивозяться на полігон твердих побутових відходів міста, також відповідно до договору з комунальним дорожньо-експлуатаційним управлінням.

Особи, винні в порушенні встановленого порядку поводження з відходами (порушення правил обліку відходів, самовільне складування і видалення відходів, передача відходів в інші підприємства/організації з порушенням встановлених правил), згідно законодавства несуть дисциплінарну, адміністративну або кримінальну відповідальність.

Висновки до розділу 4

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом, описано, які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника. Приведені рекомендації щодо організації робочого місця, а також важлива інформацію щодо пожежної та електробезпеки. Були наведені розміри приміщення та наведено значення температури, вологості й рухливості повітря, необхідна кількість і потужність ламп та інші параметри, значення яких впливає на умови праці робітника, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп’ютері.

ВИСНОВКИ

Метою дипломної роботи був аналіз та дослідження розподіленої обробки відеоданих. Також у рамках цієї роботи була розроблена модель системи збереження відеоданих для подальшої обробки.

Були проаналізовані різні методи сегментації відео для подальшої обробки. Завдяки цьому було обрано кращій метод для нашої системи - це метод сегментації на основі кількості компьютерів.

Була розроблена модель, за якою повинна функціонувати система. А також розглянуті усі її компоненти, які приймають участь у цьому. Було вирішено використовувати мікросервісну архітектуру проекту, так як це дає багато можливостей та пропонує більшу гнучкість. Також це означає, що система буде функціонувати з мінімальними втратами у випадку збою.

У свою чергу користувачі мають безпосередній доступ лише до двох частин системи. Клієнт, який встановлюється на машину користувача, дозволяє керувати файлами у контексті нашого продукту. Також користувач має доступ до веб версії системи, у якої налаштовуються принципи роботи з проектом. Ці дві частини у свою чергу вже будуть працювати з усією системою, що робить налагодження та використовування проекту дуже простим та ефективним.

Для подальшої роботи треба більш детально розглянути методи розподілення даних на кластери даних.

Результати досліджень представлені у вигляді тез на VIII Всеукраїнської науково-практичної конференції **“**Електронні апарати та системи. Проблеми створення. Перспективи розвитку**”**.

ПЕРЕЛІК ПОСИЛАНЬ

1. Distributed version control system summary – [Электронний ресурс]. Режим доступу: www/URL: http://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/ - 22.04.2016 г. – Загл. с экрана.

2. Git book – [Электронний ресурс]. Режим доступу: www/URL: https://git-scm.com/book/ - 22.04.2016 г. – Загл. с экрана.

3. The Revision Control System documentation – [Электронний ресурс]. Режим доступу: www/URL: http://www.gnu.org/software/rcs/ - 22.04.2016 г. – Загл. с экрана.

4. Diffutils – [Электронний ресурс]. Режим доступу: www/URL: https://www.gnu.org/software/diffutils/ - 22.04.2016 г. – Загл. с экрана.

5. Concurrent Versions System documentation – [Электронний ресурс]. Режим доступу: www/URL: https://savannah.nongnu.org/projects/cvs/ - 22.04.2016 г. – Загл. с экрана.

6. Apache Subversion documentation – [Электронний ресурс]. Режим доступу: www/URL: http://subversion.apache.org/ - 22.04.2016 г. – Загл. с экрана.

7. Perforce documentation – [Электронний ресурс]. Режим доступу: www/URL: https://www.perforce.com/support/documentation - 22.04.2016 г. – Загл. с экрана.

8. Bug Tracking summary – [Электронний ресурс]. Режим доступу: www/URL: http://www.dmoz.org/Computers/Software/Configuration_Management/ Bug_Tracking/ - 22.04.2016 г. – Загл. с экрана.

9. Open Source summary – [Электронний ресурс]. Режим доступу: www/URL: https://opensource.org/ - 22.04.2016 г. – Загл. с экрана.

10. Mercurial: The Definitive Guide – [Электронний ресурс]. Режим доступу: www/URL: http://hgbook.red-bean.com/ - 22.04.2016 г. – Загл. с экрана.

11. ГОСТ 12.1.044-89 ССБТ. Пожежовибухонебезпека речовин і матеріалів. Номенклатура показників і методи їх визначенняя

12. ДБН В.2.5-28:2015 Природнє і штучне освітлення

13. ДСанПіН 3.3.2.007-98 Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин

14. ДСН 3.3.6.037-99 Санітарні норми виробничого шуму, ультразвуку та інфразвуку

15. ДСН 3.3.6.042-99 Санітарні норми мікроклімату виробничих

16. НПАОП 0.00-4.12-05 Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці

17. НПАОП 0.00-4.15-98 Про розробку інструкцій з охорони праці

18. НПАОП 0.00-6.03-93 Порядок опрацювання та затвердження власником нормативних актів про охорону праці

19. НАПБ Б.03.002-2007 Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою

20. НПАОП 40.1-1.01-97 Правила безопасной эксплуатации электроустановок

21. НПАОП 40.1-1.32-01 Правила устройства электроустановок. Электрооборудование специальных установок

22. ДСН 3.3.6.039-99 Санітарні норми виробничої загальної та локальної вібрації

23. ДСТУ ГОСТ 12.1.012-90 ССБТ. Вібраційна безпека. Загальні вимоги

24. ДБН В.2.5-67:2013 Опалення, вентиляція та кондиціонування

25. ГОСТ 12.1.006-84 ССБТ. Електромагнітні поля радіочастот. Загальні вимоги безпеки. Допустимі рівні на робочих місцях і вимоги до проведення контролю

26. ГОСТ 12.1.030-81 ССБТ. Електробезпечність. Захисне заземлення. Занулення

27. НПАОП 0.00-1.28-10 Правила охорони праці під час експлуатації електронно-обчислювальних машин

ДОДАТОК А

Лістинг коду SurfDescriptor.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OpenSURFcs
{
  public class SurfDescriptor
  {

    /// <summary>
    /// Gaussian distribution with sigma = 2.5.  Used as a fast lookup
    /// </summary>
    float[,] gauss25 = new float[7, 7] {

{0.02350693969273f,0.01849121369071f,0.01239503121241f,0.00708015417522f,0.00344628101733f,0.00142945847484f,0.00050524879060f},

{0.02169964028389f,0.01706954162243f,0.01144205592615f,0.00653580605408f,0.00318131834134f,0.00131955648461f,0.00046640341759f},

{0.01706954162243f,0.01342737701584f,0.00900063997939f,0.00514124713667f,0.00250251364222f,0.00103799989504f,0.00036688592278f},

{0.01144205592615f,0.00900063997939f,0.00603330940534f,0.00344628101733f,0.00167748505986f,0.00069579213743f,0.00024593098864f},

{0.00653580605408f,0.00514124713667f,0.00344628101733f,0.00196854695367f,0.00095819467066f,0.00039744277546f,0.00014047800980f},

{0.00318131834134f,0.00250251364222f,0.00167748505986f,0.00095819467066f,0.00046640341759f,0.00019345616757f,0.00006837798818f},

{0.00131955648461f,0.00103799989504f,0.00069579213743f,0.00039744277546f,0.00019345616757f,0.00008024231247f,0.00002836202103f}
    };

    /// <summary>
    /// The integral image which is being used
    /// </summary>
    IntegralImage img;

    /// <summary>
    /// Static one-call do it all function
    /// </summary>
    /// <param name="img"></param>
    /// <param name="ipts"></param>
    /// <param name="extended"></param>
    /// <param name="upright"></param>
    public static void DecribeInterestPoints(List<IPoint> ipts, bool upright, bool extended, IntegralImage img)
```

```
    {
      SurfDescriptor des = new SurfDescriptor();
      des.DescribeInterestPoints(ipts, upright, extended, img);
    }


    /// <summary>
    /// Build descriptor vector for each interest point in the
supplied list
    /// </summary>
    /// <param name="img"></param>
    /// <param name="ipts"></param>
    /// <param name="upright"></param>
    public void DescribeInterestPoints(List<IPoint> ipts, bool
upright, bool extended, IntegralImage img)
    {
      if (ipts.Count == 0) return;
      this.img = img;

      foreach (IPoint ip in ipts)
      {
        // determine descriptor size
        if (extended) ip.descriptorLength = 128;
        else ip.descriptorLength = 64;

        // if we want rotation invariance get the orientation
        if (!upright) GetOrientation(ip);

        // Extract SURF descriptor
        GetDescriptor(ip, upright, extended);
      }
    }

    /// <summary>
    /// Determine dominant orientation for InterestPoint
    /// </summary>
    /// <param name="ip"></param>
    void GetOrientation(IPoint ip)
    {
      const byte Responses = 109;
      float[] resX = new float[Responses];
      float[] resY = new float[Responses];
      float[] Ang = new float[Responses];
      int idx = 0;
      int[] id = { 6, 5, 4, 3, 2, 1, 0, 1, 2, 3, 4, 5, 6 };

      // Get rounded InterestPoint data
      int X = (int)Math.Round(ip.x, 0);
      int Y = (int)Math.Round(ip.y, 0);
      int S = (int)Math.Round(ip.scale, 0);

      // calculate haar responses for points within radius of 6*scale
      for (int i = -6; i <= 6; ++i)
      {
        for (int j = -6; j <= 6; ++j)
```

```
        {
          if (i * i + j * j < 36)
          {
            float gauss = gauss25[id[i + 6], id[j + 6]];
            resX[idx] = gauss * img.HaarX(Y + j * S, X + i * S, 4 *
S);
            resY[idx] = gauss * img.HaarY(Y + j * S, X + i * S, 4 *
S);
            Ang[idx] = (float)GetAngle(resX[idx], resY[idx]);
            ++idx;
          }
        }
      }

      // calculate the dominant direction
      float sumX, sumY, max = 0, orientation = 0;
      float ang1, ang2;
      float pi = (float)Math.PI;

      // loop slides pi/3 window around feature point
      for (ang1 = 0; ang1 < 2 * pi; ang1 += 0.15f)
      {
        ang2 = (ang1 + pi / 3f > 2 * pi ? ang1 - 5 * pi / 3f : ang1 +
pi / 3f);
        sumX = sumY = 0;

        for (int k = 0; k < Responses; ++k)
        {
          // determine whether the point is within the window
          if (ang1 < ang2 && ang1 < Ang[k] && Ang[k] < ang2)
          {
            sumX += resX[k];
            sumY += resY[k];
          }
          else if (ang2 < ang1 &&
            ((Ang[k] > 0 && Ang[k] < ang2) || (Ang[k] > ang1 && Ang[k]
< pi)))
          {
            sumX += resX[k];
            sumY += resY[k];
          }
        }

        // if the vector produced from this window is longer than all
        // previous vectors then this forms the new dominant direction
        if (sumX * sumX + sumY * sumY > max)
        {
          // store largest orientation
          max = sumX * sumX + sumY * sumY;
          orientation = (float)GetAngle(sumX, sumY);
        }
      }

      // assign orientation of the dominant response vector
      ip.orientation = (float)orientation;
```

```
    }


    /// <summary>
    /// Construct descriptor vector for this interest point
    /// </summary>
    /// <param name="bUpright"></param>
    void GetDescriptor(IPoint ip, bool bUpright, bool bExtended)
    {
      int sample_x, sample_y, count = 0;
      int i = 0, ix = 0, j = 0, jx = 0, xs = 0, ys = 0;
      float dx, dy, mdx, mdy, co, si;
      float dx_yn, mdx_yn, dy_xn, mdy_xn;
      float gauss_s1 = 0f, gauss_s2 = 0f;
      float rx = 0f, ry = 0f, rrx = 0f, rry = 0f, len = 0f;
      float cx = -0.5f, cy = 0f; //Subregion centers for the 4x4
gaussian weighting

      // Get rounded InterestPoint data
      int X = (int)Math.Round(ip.x, 0);
      int Y = (int)Math.Round(ip.y, 0);
      int S = (int)Math.Round(ip.scale, 0);

      // Allocate descriptor memory
      ip.SetDescriptorLength(64);

      if (bUpright)
      {
        co = 1;
        si = 0;
      }
      else
      {
        co = (float)Math.Cos(ip.orientation);
        si = (float)Math.Sin(ip.orientation);
      }

      //Calculate descriptor for this interest point
      i = -8;
      while (i < 12)
      {
        j = -8;
        i = i - 4;

        cx += 1f;
        cy = -0.5f;

        while (j < 12)
        {
          cy += 1f;

          j = j - 4;

          ix = i + 5;
          jx = j + 5;
```

```
            dx = dy = mdx = mdy = 0f;
            dx_yn = mdx_yn = dy_xn = mdy_xn = 0f;

            xs = (int)Math.Round(X + (-jx * S * si + ix * S * co), 0);
            ys = (int)Math.Round(Y + (jx * S * co + ix * S * si), 0);

            // zero the responses
            dx = dy = mdx = mdy = 0f;
            dx_yn = mdx_yn = dy_xn = mdy_xn = 0f;

            for (int k = i; k < i + 9; ++k)
            {
              for (int l = j; l < j + 9; ++l)
              {
                //Get coords of sample point on the rotated axis
                sample_x = (int)Math.Round(X + (-l * S * si + k * S *
co), 0);
                sample_y = (int)Math.Round(Y + (l * S * co + k * S *
si), 0);

                //Get the gaussian weighted x and y responses
                gauss_s1 = Gaussian(xs - sample_x, ys - sample_y, 2.5f *
S);
                rx = (float)img.HaarX(sample_y, sample_x, 2 * S);
                ry = (float)img.HaarY(sample_y, sample_x, 2 * S);

                //Get the gaussian weighted x and y responses on rotated
axis
                rrx = gauss_s1 * (-rx * si + ry * co);
                rry = gauss_s1 * (rx * co + ry * si);


                if (bExtended)
                {
                  // split x responses for different signs of y
                  if (rry >= 0)
                  {
                    dx += rrx;
                    mdx += Math.Abs(rrx);
                  }
                  else
                  {
                    dx_yn += rrx;
                    mdx_yn += Math.Abs(rrx);
                  }

                  // split y responses for different signs of x
                  if (rrx >= 0)
                  {
                    dy += rry;
                    mdy += Math.Abs(rry);
                  }
                  else
                  {
```

```
                    dy_xn += rry;
                    mdy_xn += Math.Abs(rry);
                  }
                }
                else
                {
                  dx += rrx;
                  dy += rry;
                  mdx += Math.Abs(rrx);
                  mdy += Math.Abs(rry);
                }
              }
            }

            //Add the values to the descriptor vector
            gauss_s2 = Gaussian(cx - 2f, cy - 2f, 1.5f);

            ip.descriptor[count++] = dx * gauss_s2;
            ip.descriptor[count++] = dy * gauss_s2;
            ip.descriptor[count++] = mdx * gauss_s2;
            ip.descriptor[count++] = mdy * gauss_s2;

            // add the extended components
            if (bExtended)
            {
              ip.descriptor[count++] = dx_yn * gauss_s2;
              ip.descriptor[count++] = dy_xn * gauss_s2;
              ip.descriptor[count++] = mdx_yn * gauss_s2;
              ip.descriptor[count++] = mdy_xn * gauss_s2;
            }

            len += (dx * dx + dy * dy + mdx * mdx + mdy * mdy
                    + dx_yn + dy_xn + mdx_yn + mdy_xn) * gauss_s2 *
gauss_s2;

            j += 9;
          }
          i += 9;
        }

        //Convert to Unit Vector
        len = (float)Math.Sqrt((double)len);
        if (len > 0)
        {
          for (int d = 0; d < ip.descriptorLength; ++d)
          {
            ip.descriptor[d] /= len;
          }
        }
      }


      /// <summary>
      /// Get the angle formed by the vector [x,y]
      /// </summary>
```

```
    /// <param name="X"></param>
    /// <param name="Y"></param>
    /// <returns></returns>
    double GetAngle(float X, float Y)
    {
      if (X >= 0 && Y >= 0)
        return Math.Atan(Y / X);
      else if (X < 0 && Y >= 0)
        return Math.PI - Math.Atan(-Y / X);
      else if (X < 0 && Y < 0)
        return Math.PI + Math.Atan(Y / X);
      else if (X >= 0 && Y < 0)
        return 2 * Math.PI - Math.Atan(-Y / X);

      return 0;
    }


    /// <summary>
    /// Get the value of the gaussian with std dev sigma
    /// at the point (x,y)
    /// </summary>
    /// <param name="x"></param>
    /// <param name="y"></param>
    /// <param name="sig"></param>
    /// <returns></returns>
    float Gaussian(int x, int y, float sig)
    {
      float pi = (float)Math.PI;
      return (1f / (2f * pi * sig * sig)) * (float)Math.Exp(-(x * x +
y * y) / (2.0f * sig * sig));
    }


    /// <summary>
    /// Get the value of the gaussian with std dev sigma
    /// at the point (x,y)
    /// </summary>
    /// <param name="x"></param>
    /// <param name="y"></param>
    /// <param name="sig"></param>
    /// <returns></returns>
    float Gaussian(float x, float y, float sig)
    {
      float pi = (float)Math.PI;
      return 1f / (2f * pi * sig * sig) * (float)Math.Exp(-(x * x + y
* y) / (2.0f * sig * sig));
    }


  } // SurfDescriptor
} // OpenSURFcs
```

ДОДАТОК Б

Комп'ютерна презентація

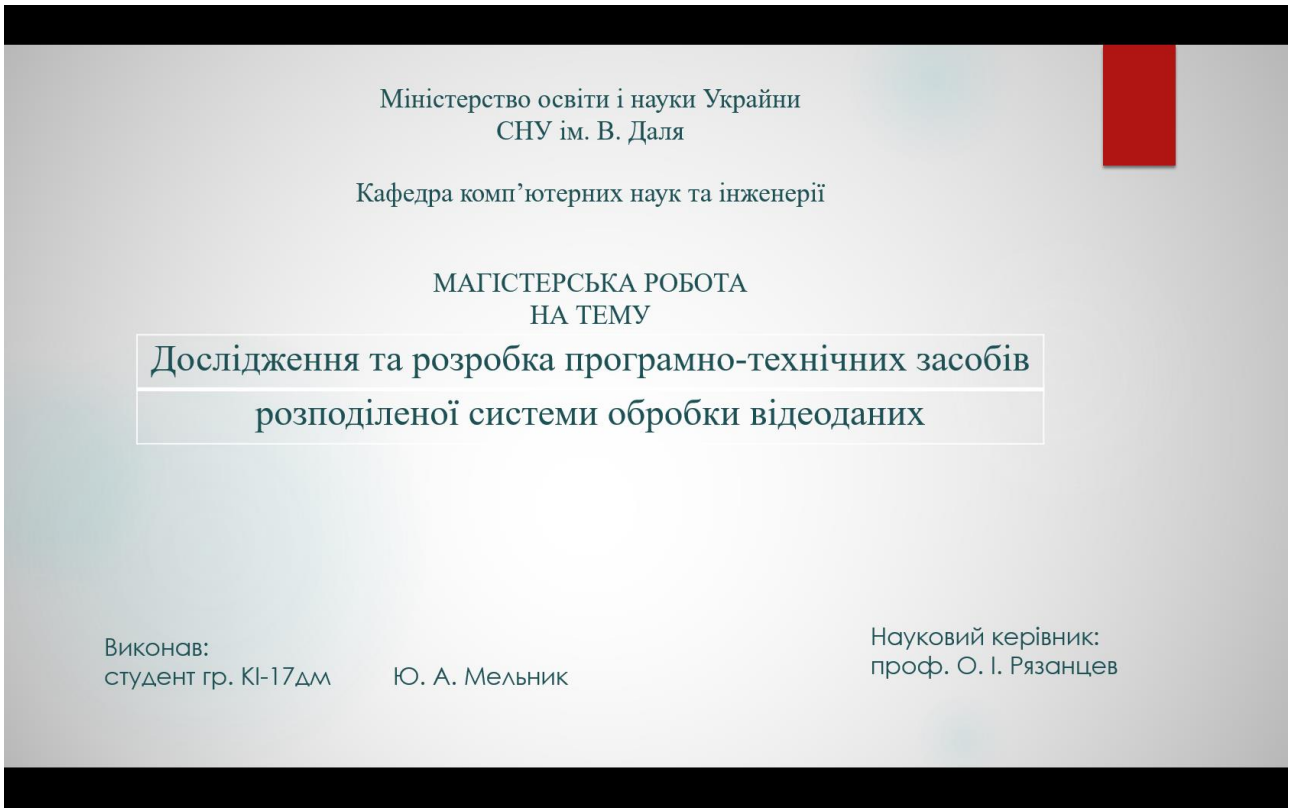

Рисунок Б.1- Слайд №1

Актуальність теми

В останні роки ми спостерігаємо істотне збільшення розширень відео, що знімаються споживчою електронікою, і сучасні комп'ютери не можуть впоратися з цим. Сучасні камери знімають відео в розширенні до 8к, що робить обробку відео дуже тривалим процесом. Наприклад, обробка 4х хвилинного 8к відео на найсучаснішому комп'ютері займе приблизно 4 години.

Рисунок Б.2- Слайд №2

Постановка задачі

- Мета роботи - дослідити можливість розподіленої обробки відеоданих. Для досягнення поставленої мети потрібно вирішити наступні завдання:
- - дослідження моделі MapReduce для розподіленої обробки даних;
- - дослідження схем сегментації відео для подальшої обробки;
- - розробка програмної реалізації моделі.

Рисунок Б.3- Слайд №3

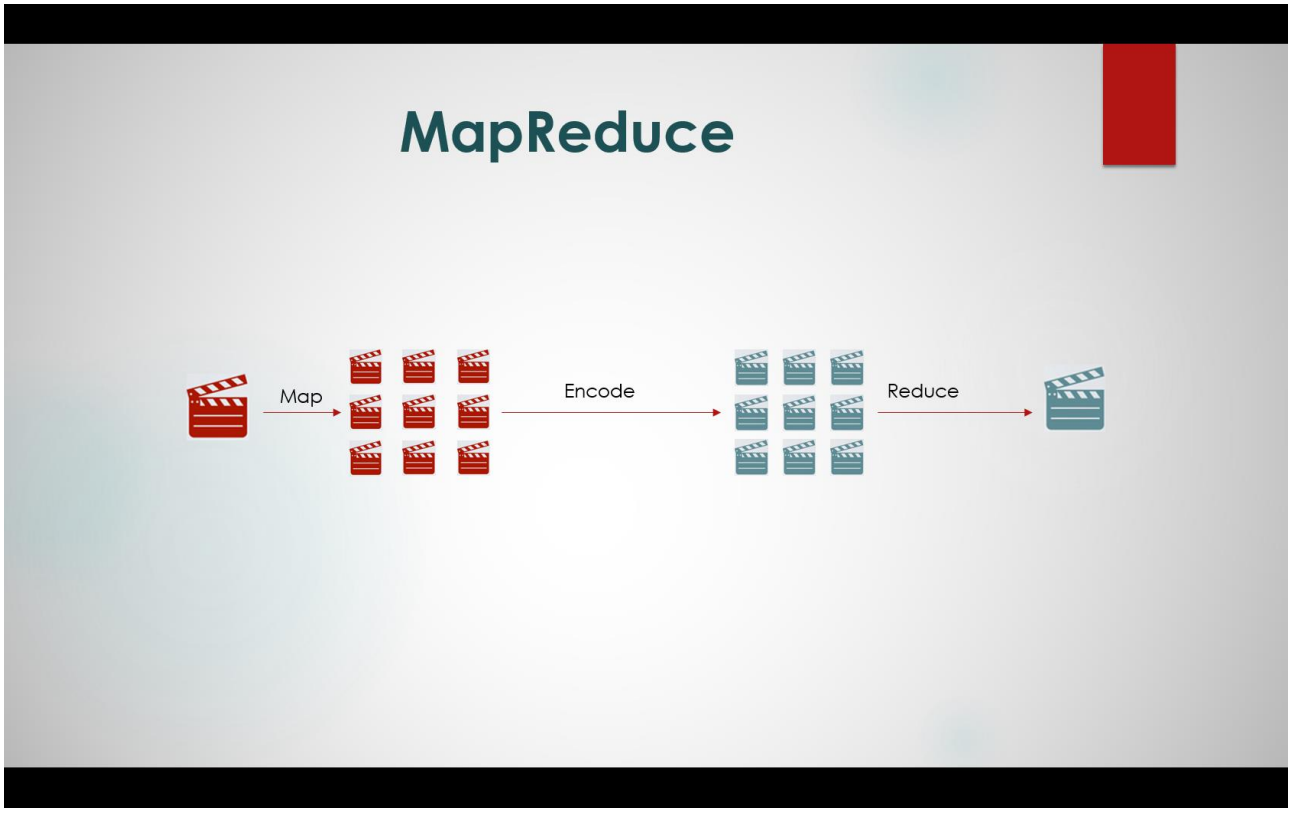

Рисунок Б.4- Слайд №4

Рисунок Б.5- Слайд №5

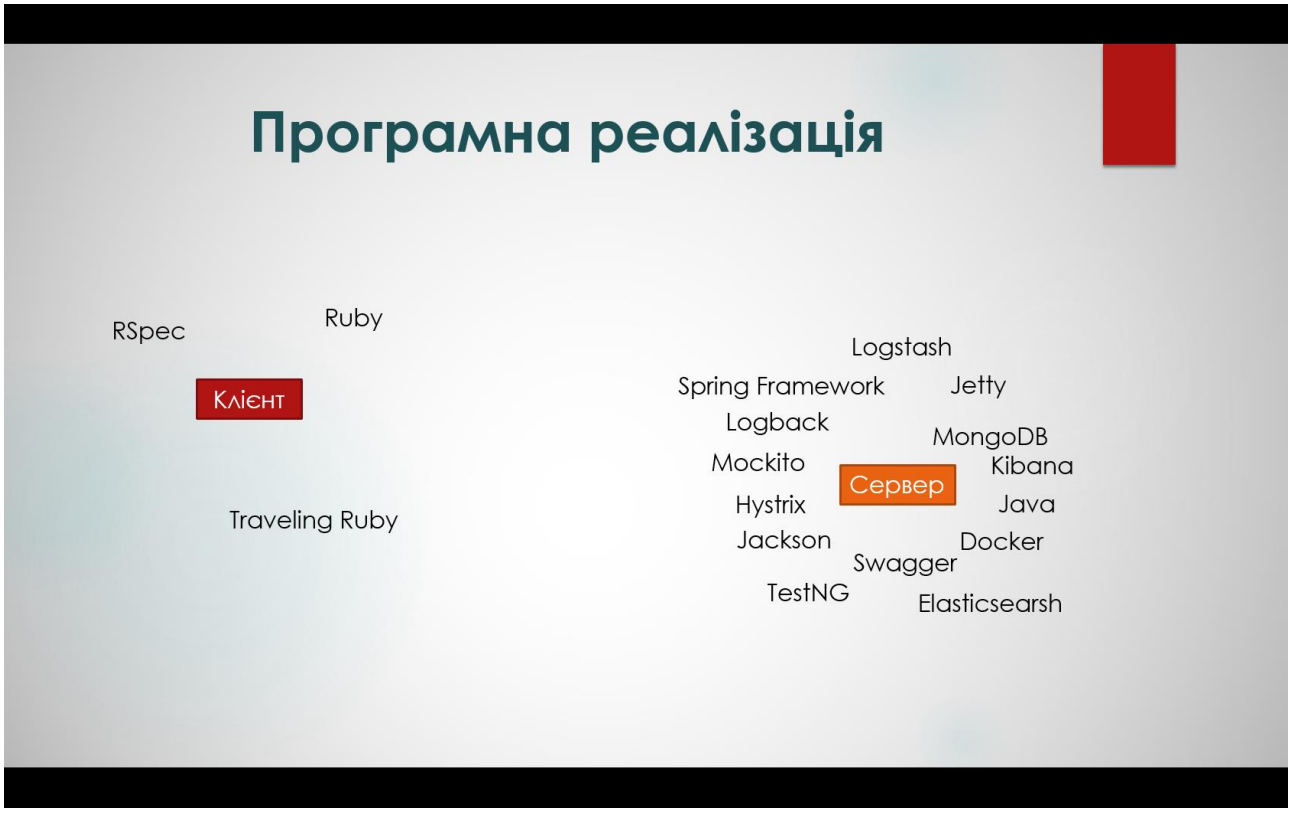

Рисунок Б.6- Слайд №6

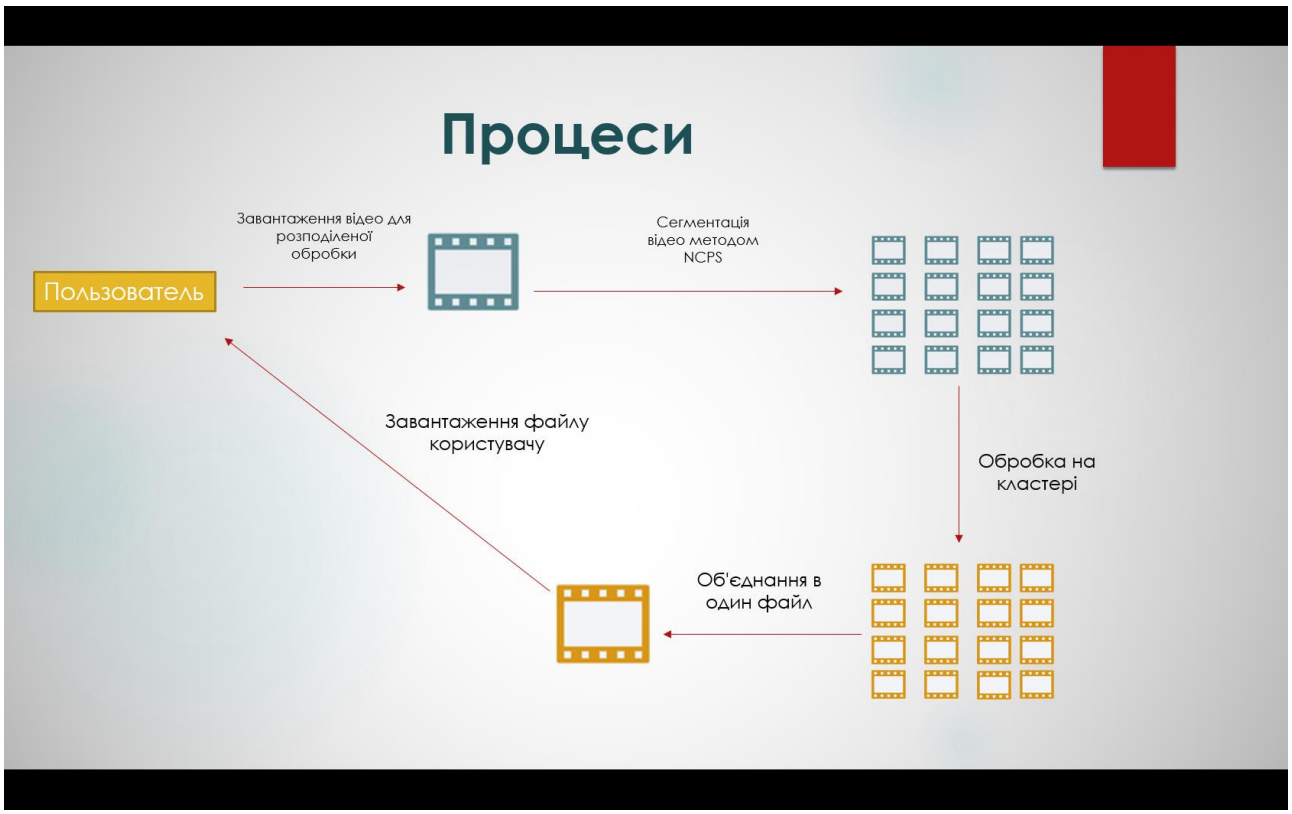

Рисунок Б.7- Слайд №7

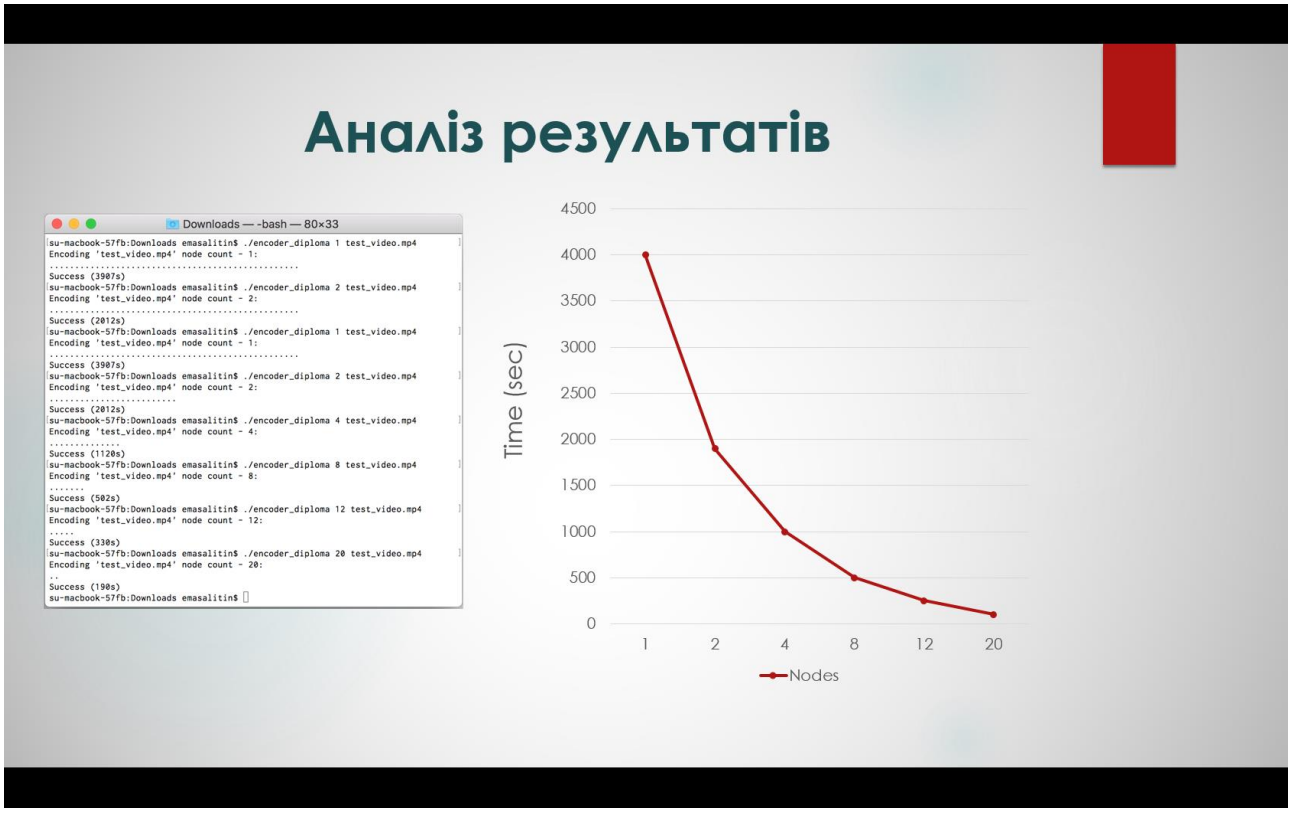

Рисунок Б.8- Слайд №8

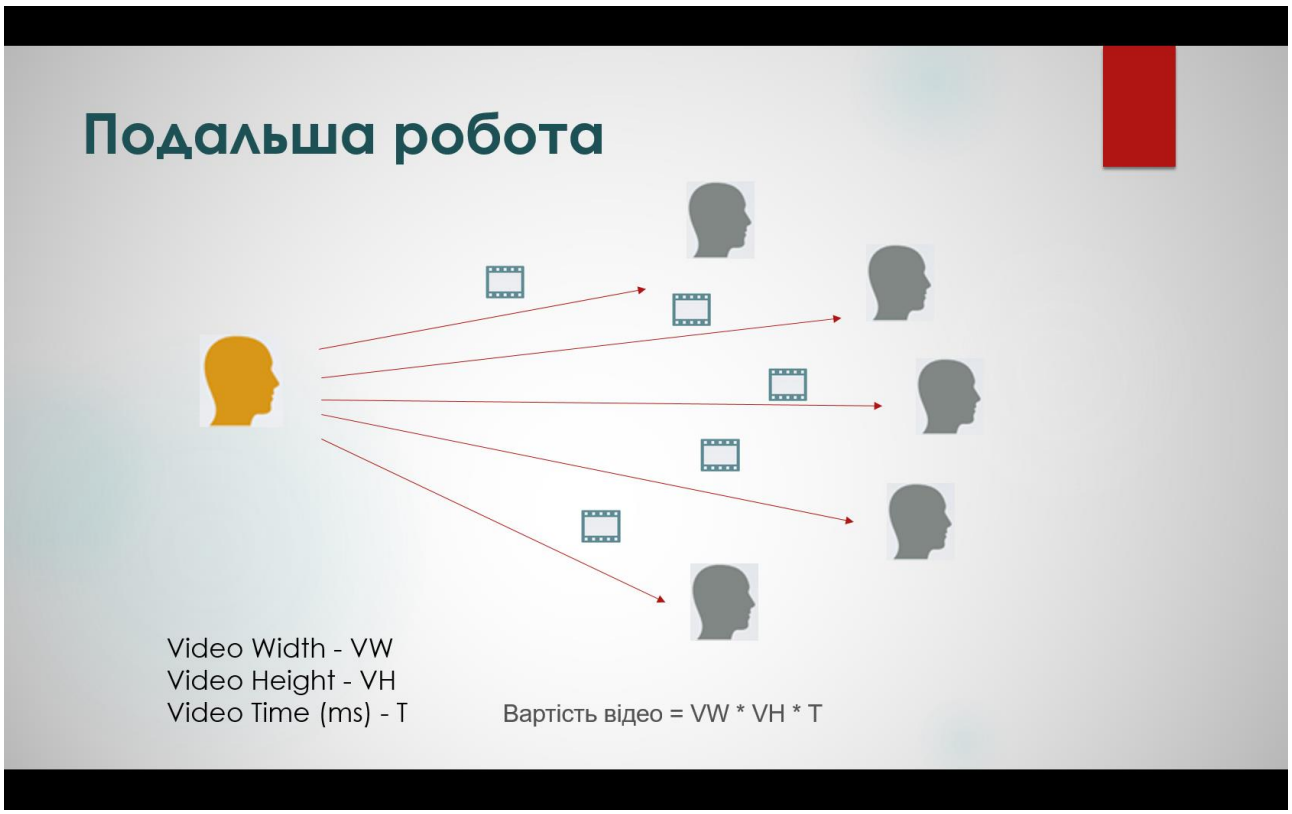

Рисунок Б.9- Слайд №9

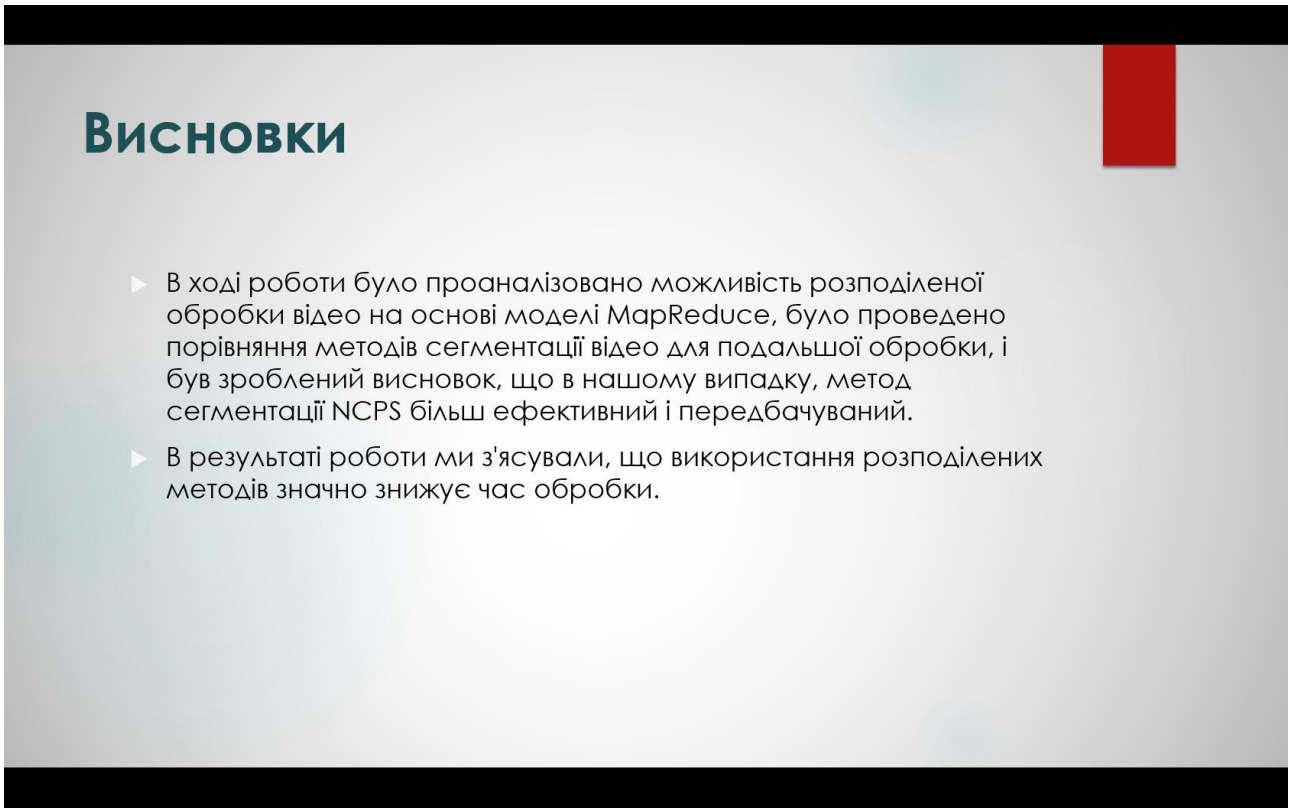

Рисунок Б.10- Слайд №10