МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

кафедра «Інформаційні системи та мережі»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи на тему:

Інформаційна система розподіленого зберігання файлів

Студента групи **КН-411 Нанівського О.І.**
(шифр, прізвище та ініціали)

Керівник роботи Рішняк І.В. (_____)

Консультант Гринів Т.Т. (_____)

Нормоконтроль Василюк А.С. (_____)

Завідувач кафедри ІСМ Литвин В. В.

«_28__» __травня_____ **2021 р.**

ЛЬВІВ – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра «Інформаційні системи та мережі»

Спеціальність 122 "Комп'ютерні науки"

Перший (бакалаврський) рівень вищої освіти

«ЗАТВЕРДЖУЮ»

Завідувач кафедри ІСМ Литвин В.В.

«_28_» __травня___ 2021 р.

ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу студента групи КН-411

Нанівського Олега Ігоровича

(прізвище, ім’я, по батькові)

1. Тема роботи “Інформаційна система розподіленого зберігання файлів”

затверджена наказом по НУ «ЛП» від «_10_» __березня___2021р. №_652-4-08_____

2. Термін здачі студентом закінченої роботи 21.05.2021

3. Вихідні дані для роботи: DF-діаграми, інструкція користувача

4.Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити): проаналізувати літературні джерела, провести системний аналіз, описати програмні засоби, показати практичну реалізацію, провести економічні розрахунки.

5. Перелік графічного матеріалу: діаграми IDEF0, приклади виконання програми в терміналі, архітектури досліджуваних систем, статистика вибраних програмних засобів, інтерфейс VSCode.

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту): VSCode, Python, Linux.

7. Консультування роботи, із зазначенням розділів роботи

Розділ	Консультанти	Підпис, дата	
		завдання видав	завдання отримав
Економічний	Гринів Т.Т.	25.03.2021	25.03.2021

8. Дата, коли видано завдання _____1.02.2021 р._____

Керівник _____
(підпис)

Завдання отримав до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Етапи бакалаврської кваліфікаційної роботи	Термін виконання етапів роботи	Примітки
1	Вступ	24.03.2021	Виконано
2	Дослідження літературних джерел	31.03.2021	Виконано
3	Систем аналіз	07.04.2021	Виконано
4	Опис програмних засобів	14.04.2021	Виконано
5	Практична реалізація	21.04.2021	Виконано
6	Економічний аналіз	28.04.2021	Виконано
7	Висновок	05.05.2021	Виконано
8			
9			

Студент _____
(підпис)

Керівник роботи _____
(підпис)

ЗМІСТ

ВСТУП

Розподілені системи для зберігання файлів надають можливості зручного та безпечного зберігання використовуючи віддалені машини. Починаючи від їх першої імплементації NFS, яка комерційно використовувалася, розподілені файлові системи постійно вдосконалювалися і набували ще більшої популярності. Сьогодні вони є незамінними в сфері великих даних і хмарних обчислень.

Актуальність теми. Великий попит на розподілені файлові системи призвів до зростання пропозицій на ринку. Проте всього кілька компаній таких, як Google, AWS, Oracle разом займають більшу частину ринку, що не дозволяє іншим реалізаціям набути популярності, окрім локального ринку. Також враховуючи масштаб компаній, фокус їхньої уваги може легко обійти локальний ринок. Так розробка вітчизняного аналога зможе надавати кращі послуги орієнтовані на українські компанії.

Мета дослідження. Дана робота має на меті продемонструвати принципи роботи розподілених файлових систем. Вона може стати основою для майбутнього комерційного проекту, або використана для навчальних цілей.

Для її розробки необхідно розв'язати такі завдання:

- Провести аналіз предметної області. Дослідити відомі реалізації подібних систем.
- Побудувати архітектуру системи, забезпечивши прозорість системи, клієнт не знає про структуру файлової системи, а бачить лише один простір імен, а також надмірність системи, що забезпечить її толерантність до відмов. Провести системний аналіз.
- Проаналізувати технічні засоби за допомогою, яких можна побудувати систему.
- Реалізувати систему та провести її тестування.

Об'єктом дослідження є процес безпечного зберігання та читання файлів в розподіленій файловій системі.

Предметом дослідження методи та засоби зберігання і читання файлів з розподіленої файлової системи.

Практичне значення. Створена система встановлена на віддалений сервер дозволяє легко і надійно зберігати і читати файли в будь який момент з будь якого місця за умови доступу до інтернету.

РОЗДІЛ 1

Аналіз джерел

1.1. Архітектурні принципи розподілений файлових систем

Розподілена файлова система – це система, яка розподілена на багатьох серверах. Це дозволяє програмам отримувати або зберігати файли як локальні з будь-якого комп'ютера або мережі.

Головна мета розподіленої файлової системи – це дозволити фізично віддаленим користувачам ділитися їхніми даними і ресурсами використовуючи звичайну файлову систему.

Розподілена файлова система має два компоненти [2]:

- Прозорість місце знаходження, яке здійснюється через простір імен.
- Надмірність використовуючи реплікацію файлів.

Прозорість приносить зручність для кінцевого користувача даючи йому змогу маніпулювати, читати і зберігати файли на локальній машині в той час, як сам процес відбувається на серверах.

В загальному розподілені файлові системи використовуються в LAN мережах, але вони також можуть бути використаними в WAN [17].

Окрім розподіленої файлової системи іншим варіантом віддаленого доступу до файлів є ділитися диском. Ця система дає клієнту контроль доступу, що призводить до недоступності файлів коли клієнт офлайн. Розподілена файлова система в свою чергу є толерантною до помилок і клієнт має доступ до даних навіть якщо, декілька вузлів системи є недоступними [3].

Перевагами розподіленої файлової системи можна назвати обмеження в доступі базуючись на списку доступу або можливостей сервера і клієнта спираючись на то, як розроблено протокол. Оскільки сервер також надає одну центральну точку доступу припускається, що він буде справно працювати попри помилку окремих вузлів, як показано на Рис.1.1. Проте єдина точка доступу також і є його слабкістю. Наприклад сервер може перестати працювати через DDoS атаку.

Тому зазвичай використовують запасний сервер для резервного збереження даних [3].

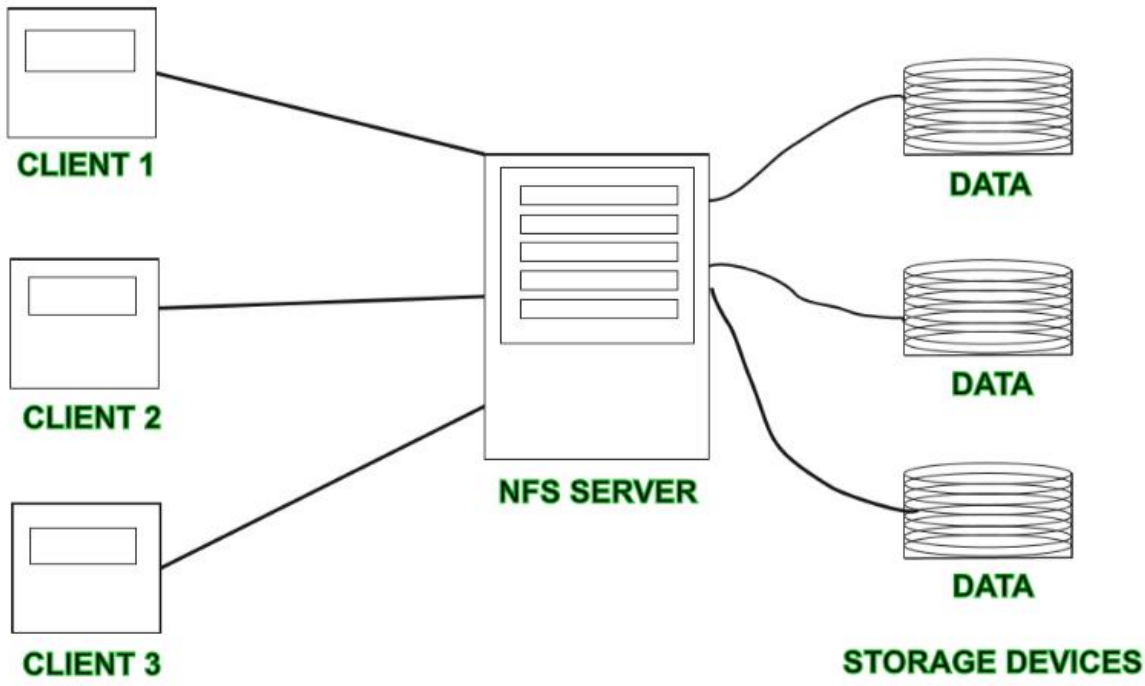

Рис.1.1 Абстрактна архітектура розподіленої файлової системи

Розглянемо короткий огляд основних властивостей розподіленої файлової системи [2]:

- Прозорість
  - Прозорість структури: клієнт не знає про кількість або розміщення файлових серверів. Велика кількість файлових серверів використовується для збільшення продуктивності, адаптивності та надійності.
  - Прозорість доступу: отримання файлу на локальній і віддалених машинах не має відрізнятися, система автоматично знаходить відповідний файл і надсилає його клієнту.
  - Прозорість імен: ім’я файлу не має змінюватися, або вказувати на його місце знаходження.
  - Прозорість реплікації: якщо файл скопійований на декількох вузлах, його копії і місце знаходження повинні бути сховані від інших вузлів.
- Мобільність користувача: система автоматично приносить домашню директорію користувача до вузла з якого користувач здійснює вхід.

- Продуктивність: продуктивність визначається середнім часом для обробки запиту клієнта. Рекомендовано, щоб цей час був подібним до централізованих систем.
- Простота: інтерфейс має бути простим з невеликою кількістю команд.
- Висока доступність: система повинна працювати в разі часткових помилок, як помилка з'єднання або помилка вузла. Високоадаптивні розподілені файлові системи повинні мати різні і незалежні файлові сервера для контролю різних і незалежних пристроїв зберігання.

1.2. Аналіз відомих реалізацій

Причиною популярності розподілених файлових систем стала простота інформаційного обміну між багатьма користувачами. Також їхніми основними особливостями мають бути: узгодженість даних, рівномірність доступу, надійність, ефективність, продуктивність, керованість, доступність та безпека [4].

Існує багато розподілених файлових систем, деякі з них є CephFS, LustreFS, GlusterFS, XtreemFS, Plane 9, GPFS, GFS, та MooseFS.

1.2.1. Network File System — NFS

Розроблена в 1984р. компанією Sun Microsystems. Це одна з найперших розподілених файлових систем, яку широко застосовували. Сьогодні нею керує Internet Engineering Task Force. Остання версія NFSv4.2 була затверджена в листопаді 2016 і вимагає використання TCP протоколу, коли 2 і 3 версії дозволяли UDP протокол [5].

Архітектура NFS досить проста (Рис.1.2), проте це також накладає деякі обмеження. NFS була розроблена для прямого доступу до єдиного місця збереження даних на віддаленій машині. Також, для збільшення продуктивності, система зберігає кеш даних та метаданих на машині клієнта, це дозволяє зменшити час для наступних запитів [6].

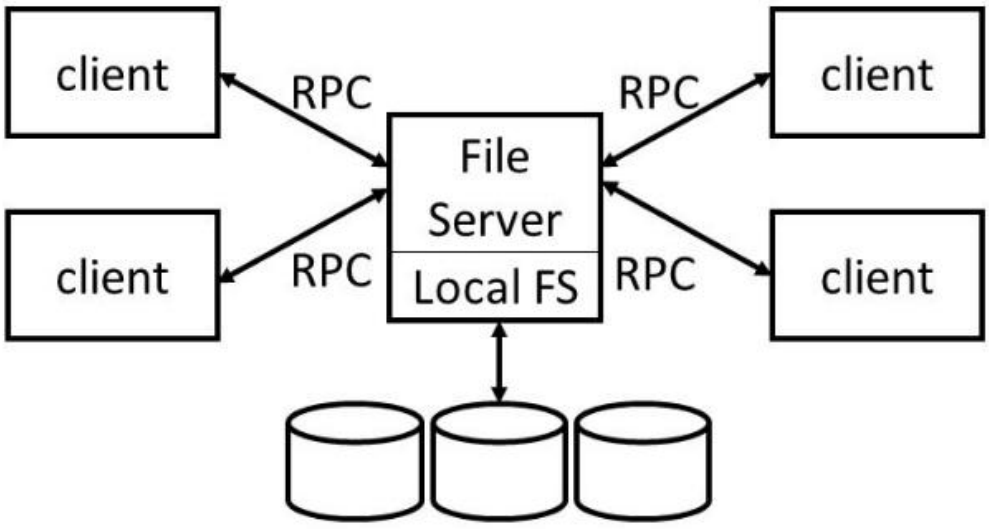

Рис.1.2 Архітектура NFS

Обмеженнями такої системи є брак надійності, оскільки усі файли знаходять на одній машині, помилка на цій машині приведе до недоступності файлів для користувачів. Іншим обмеженням є те, що NFS легко перевантажується, при доступі великої кількості користувачів [4].

Перевагою NFS є низька вартість встановлення, яку легко встановити, адже вона використовує наявну IP інфраструктуру. Також значною перевагою є можливість центрального керування, зменшуючи потребу в додатковому програмному забезпечені на системі користувача. Система є прозорою, що дає змогу клієнтам отримати доступ наче до локального жорсткого диску [5].

Отже, NFS призначена для невеликого масштабу, де є невелика кількість користувачів, яка одночасно звертається до серверу.

1.2.2. Andrew file System — AFS

Вперше представлена в 1980 роках університетом Карнегі Меллон для вирішення проблеми масштабованості серед розподілених файлових систем [4].

В свій час AFS була досить популярною системою. Вона використовувалася близько 20,000 клієнтами в 10 країнах світу. Оцінюється, що вона мала більше, ніж 100,000 користувачів [7].

В початковій версії AFS, цілий файл кешувався на пристрої користувача, щоб збільшити продуктивність для запитів для одного файлу на сервері (Рис.1.3). Що дозволило знизити завантаженість сервера, який містить цей файл. Вкінці, якщо файл був модифікований на локальній системі, остання версія надсилалася на сервер під час закриття сесії. Проте, перший доступ до не кешованого файлу був неефективним [4].

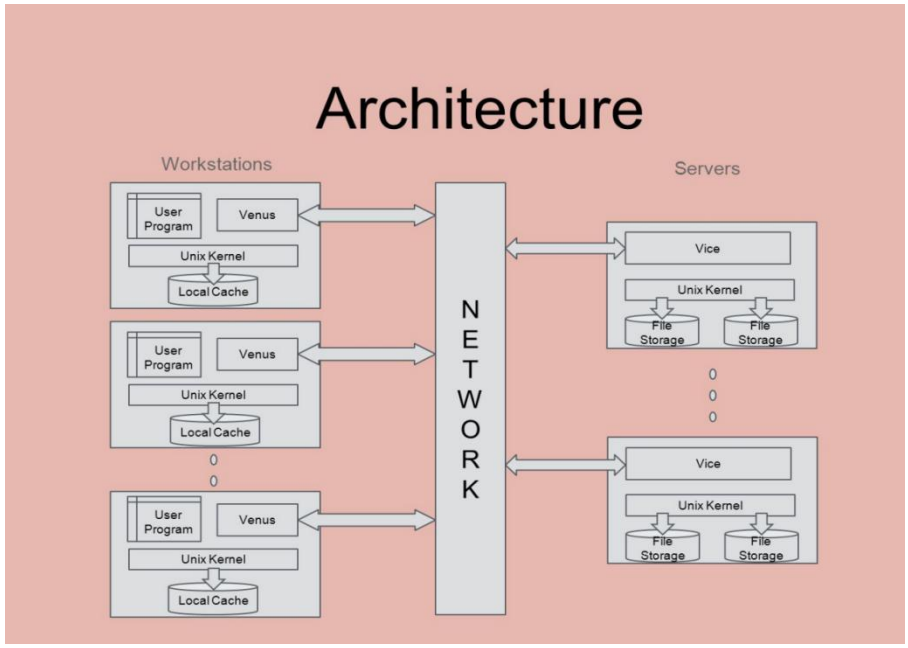

Рис.1.3 Архітектура AFS

Обмеженнями даної системи є [4]:

1. Високий час проходження шляху: щоб, отримати доступ до файлу на стороні клієнта, сервер повинен пройти повний шлях з домашньої директорії до місце знаходження файлу. Цей процес займає значні ресурси сервера, які він міг витратити на обробку запитів від клієнтів.

2. Високий трафік: в AFS значна частина трафіку створюється у вигляді повідомлень перевірки чи файл був модифікований чи ні. Цей трафік зменшує ефективність в мережі.

Щоб покращити початкову версію AFS, було запропоновано модифікації для масштабованості та продуктивності [4]:

1. Використання ідентифікатора файлів: цей ідентифікатор чітко вказує, який файловий сервер зацікавлений, що зменшує завантаженість головного сервера.

2. Використання функцій зворотних викликів: ця проста функція виконується сервером, яка повідомляє про модифікацію файлу до всіх клієнтів, які мають кешовану немодифіковану версію цього файлу. Тому, звичайні повідомлення перевірки більше непотрібні.

Andrew file System мала значний вплив на сучасні середовища хмарних обчислень своєю моделлю зберігання даних в хмарі і доставляння частин за допомогою кешування за вимогою. Це спрощує керування операціями зберігаючи продуктивність та масштабованість для кінцевих користувачів. Всі дані, які відносяться до певного користувача доставляються через мережу за вимогою, зберігаючи синхронізованими всі машини [8].

Таблиця 1.1

Порівняльний аналіз NFS та AFS

Критерій	**NFS**	**AFS**
Простір імен	Не має простору імен спільного користування, індивідуальні простори імен для всіх клієнтів.	Глобальний простір імен спільного користування.
Кешування файлів	Не має локального кешування файлів.	Файли повністю кешуються на локальному диску.
Масштабованість	Для невеликої кількості користувачів (10-20).	Порівняно з NFS високо масштабована.
Безпека	ІД користувачів використовується для визначення доступу до файлу.	Протокол kerberos для верифікації.
Резервне копіювання	Основане на резервному копіюванні UNIX системи.	Має власну систему резервного копіювання.
Впровадження	Solaris, AIX, FreeBSD	Transarc (IBM), OpenAFS

1.2.3. Google File system — GFS

Хоча наступна версія AFS ввела значні покращення, Google все ж стикалася з проблемою зберігання і обробки великих даних. В 2003 році, вони розробили Google File System, як їхнє власне вирішення проблеми керування великими даними. Це розподілена масштабована файлова система, розроблена для підтримки великих розподілених додатків з великою кількістю даних. Вона була розроблена з багатьма цілями спільними з іншими розподіленими файловими системами. Проте, були також інші фактори, які мотивували розробників, такі як, поламка апаратного забезпечення, обмеження пропускної здатності та тип затримки [4].

Властивостями GSF є: відмовостійкість, реплікація даних, автоматичне відновлення даних, висока пропускна здатність, зменшена взаємодія між клієнтом і основним сервером, керування простором імен, висока доступність.

Найбільший GFS кластер має більше ніж 1,000 вузлів з 300 терабайт дискового простору, що може бути доступний для сотень клієнтів [9].

Розробка Google File System спрямовувалася певними припущеннями, які були виведені з попередніх спостережень [1]:

- Система побудована з багатьох недорогих компонентів, які часто ламаються. Вона повинна постійно та періодично сканувати себе, і виявляти, відновлювати від помилок компонентів.
- Система зберігає скромну кількість великих файлів. Очікується зберігання мільйонів файлів від 100 МБ. Великі файли на кілька гігабайт є звичним випадком, тому повинні оброблятися ефективно. Також, повинна бути підтримка малих файлів, проте не потрібно проводити оптимізацію для них.
- Завантаженість в основному складається з двох видів читань:
  - великі потокові читання: одна операція часто читає сотні кілобайт, частіше мегабайт, або більше. Успішні операції від одного клієнта зазвичай читають близькі регіони файлу.
  - малі випадкові читання: читають кілька кілобайт з довільним зміщенням.

- Завантаженість також має багато послідовних записів, що додають дані до файлів. Типові розміри операцій схожі для читання. Після запису, файл рідко модифікується. Малі записи на довільні позиції мають підтримуватися, але не повинні бути ефективними.
- Система повинна ефективно впроваджувати добре визначену семантику для паралельного додавання до одного й того самого файлу багатьма користувачами.
- Висока пропускна здатність важливіша за низьку затримку.

Як показана на Рис. 1.4 GFS кластер складається з одного сервера (master) та багатьох серверів даних (chunkservers), які доступні багатьом клієнтам одночасно. Вона працює на Linux машинах, як сервер процес. Досить легко запустити сервери даних і клієнта на одній машині допоки ресурси машини дозволяють. Файли поділені на чанки фіксованих розмірів, кожним з них керує основний сервер. Всі чанки зберігаються на локальних дисках, як файли для запису і читання даних.

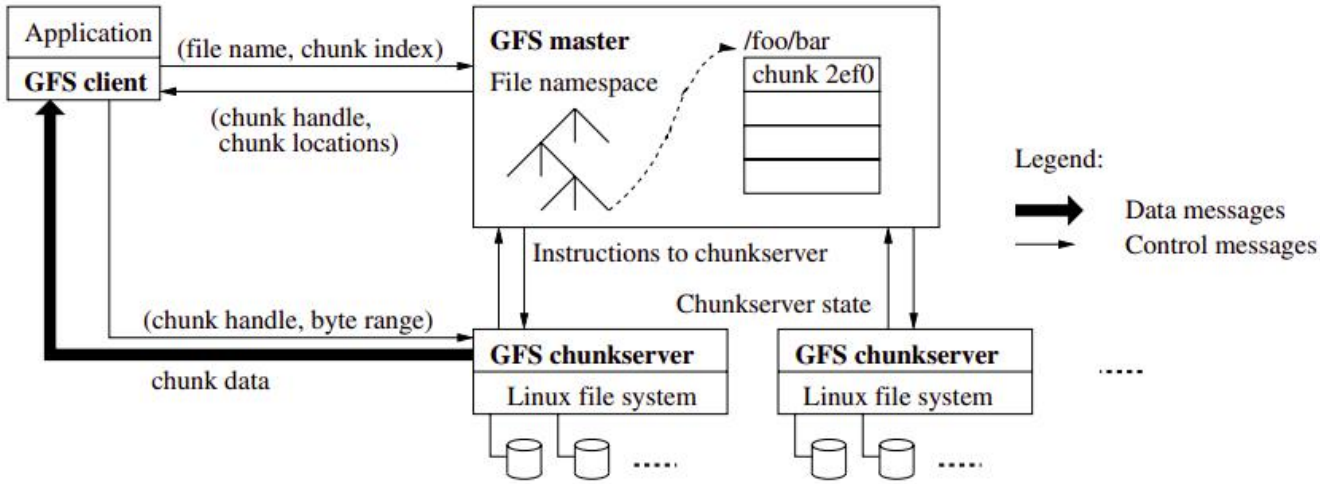

Рис. 1.4 Архітектура GFS

Для надійності кожний чанк дублюється на кількох серверах даних. Основний сервер зберігає усі метадані файлової системи, що включає простір імен і інформацію про контроль доступу, як словник чанків та їхнього розташування. Основний сервер періодично комунікує з окремими чанками для перевірки їхнього

стану, або надати їх інструкції. Для операцій пов'язаних з метаданими клієнт комунікує з основним сервером напряму. Для передачі даних файлів клієнт комунікує з серверами даних напряму, не включаючи основний сервер, що дозволяє зняти з нього завантаження [4].

1.2.4. Hadoop Distributed File System - HDFS

HDFS надихалася можливостями GFS працювати на дешевих компонентах. На відміну від інших розподілених файлових систем, HDFS вважається високонадійною розподіленою файловою системою. Вона була розроблена використовуючи дешеве обладнання, яке може зберігати велику кількість даних і забезпечити простіший доступ до них. Щоб зберігати такі великі чанки даних, файли зберігаються на багатьох машинах. Ці файли зберігаються в надлишковому форматі, щоб зберегти їх від можливої втрати в випадку поламки машини [4].

Hadoop є проектом верхнього рівня організації Apache Software Foundation, тому центральним репозиторієм рахується Apache Hadoop, проте її реалізації також є в Cloudera, Hortonworks та MapR [10].

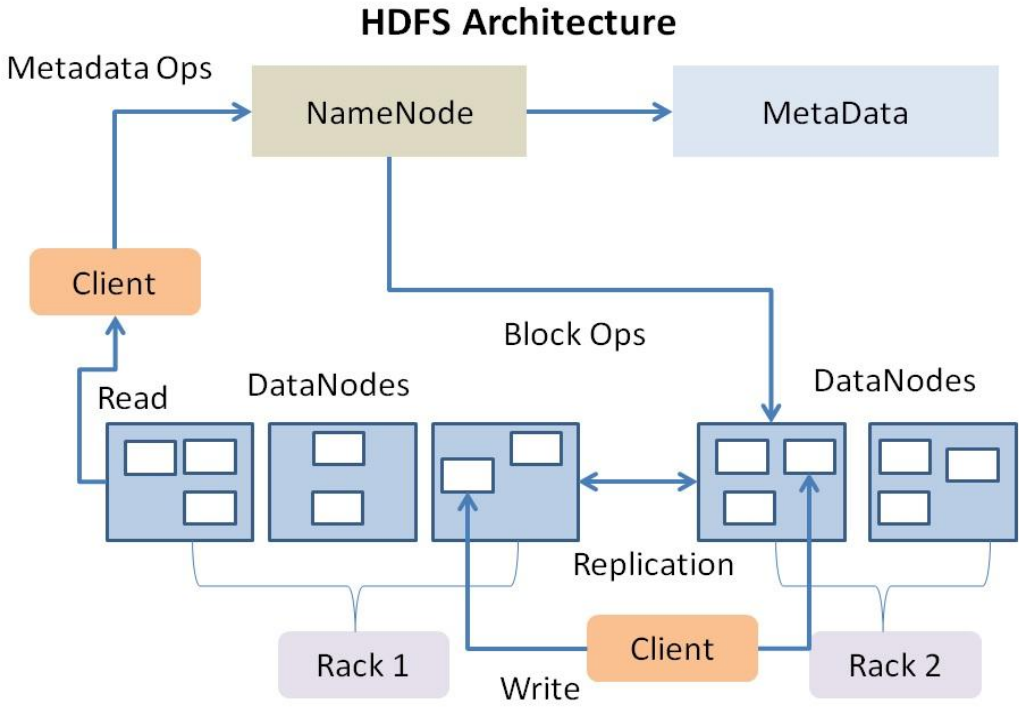

Рис. 1.5 Архітектура HDFS

Hadoop Distributed File System має master-slave архітектуру NameNode-DataNode, подібну до архітектури GFS (Рис. 1.5). Кластер складається з одного NameNode, основного серверу, який керує простір імен файлової системи і регулює доступ до файлів, який відбуваються клієнтом. Також в кластері є кілька DataNode, зазвичай один на вузол.

Таблиця 1.2

Порівняльний аналіз GFS та HDFS

Критерій оцінки	**GFS**	**HDFS**
Апаратне забезпечення	Дешеві компоненти з Linux системою, які мають високий шанс поламки	Дешеві компоненти з великою місткістю
Комунікація	TCP комунікація Періодичні командні повідомлення для синхронізації	RPC поверх TCP/IP Періодичні командні повідомлення для синхронізації
Ієрархія	MasterNode і ChunkNodes	NameNode і DataNodes
Безпека	Google має багато серверів даних в невідомих місцях, щоб збільшити надмірність	Безпека основана на POSIX моделі користувачів і груп
Розмір блоку	64 мегабайти	128 мегабайт
Метадані для блоку	Кожен чанк має 64КБ даних і 32 біти чек суми	Для кожного чанку створено два файли: 1. Файл даних 2. Файл метаданих
Фактор реплікації	3 рази, але може бути встановлена будь-яка кількість	2 або 3 рази
Хто використовує	Google Inc.	Yahoo!, Facebook, IBM
Операції	Можливі випадкові записи, або читання Модель багатьох читань і записів	Можливе тільки додавання до файлу Багато читань, одинарний запис

Отже, HDFS показує високу масштабовану і толерантну до помилок файлову систему, яка дозволяє ефективно зберігати дані в файлах. Внутрішньо, файл

розподілений на багато блоків, які зберігаються на кількох DataNodes, щоб забезпечити живучість даних в разі помилок. NameNode відповідальний за виконання операції простору імен файлової системи, таких як, відкриття, перейменування, закриття файлів чи директорій. Розподілення блоків до серверів даних також виконується головним сервером. Операції клієнта читання або запису виконуються серверами даних. Сервери даних також допомагають в створені, перейменуванні, або видалені блоків, як інструкції головного серверу [4].

1.2.5. Обмеження і покращення HDFS і GFS

Google File System задумана зберігати великі чистина даних в надійний спосіб і хоча вона підтримує глобальний простір імен, ефективну обробку, знімки простору імен і одинарний запис, система має певні обмеження:

- Випадкові записи на малому рівні: основним завантаженням GFS є великі потокові записи. Забравши певний функціонал з GFS, ми можемо отримати більше переваг для випадкових записів в малі файли.
- Один розмір чанків: деколи єдиний розмір чанку в 64 мегабайти може створювати проблеми. Наприклад, коли останній чанк поділений на кілька частин для пошуку нового місця зберігання. Це також призводить до перевикористання комунікації.

Обидві GFS і HDFS не призначені для [4]:

1. Доступу до даних з низькою затримкою: оскільки дані зберігаються великими частинами обидві розподілені файлові системи відстають в провадженні швидшого доступу до файлів. Обидві створені для передачі великих частин даних для більшої пропускної здатності.
2. Обробка файлів невеликого розміру: обидві розподілені файлові системи не призначені для обробки файлів менше 1 мегабайта. Хоча вони підтримують операції з малими файлами це призводить до втрати ефективності.

3. Випадки частої зміни даних: розподілені файлові системи добре справляються з великими одиничними записами, проте стикаються з проблемами при частій зміні даних.

Для вирішення проблеми низької затримки доступу до даних можна використати HBase. HBase — це розподілена, відсортована карта за зразком великої таблиці Google для швидших запитів в великих кількостях структурованих даних. Це база даних з відкритим кодом і масштабованістю на основі Hadoop. HBase працює на HDFS і написана на Java. HDFS файли індексуються в HBase для швидких запитів (Рис. 1.6). Головною метою для HDFS є групова обробка файлів з великим розміром. Читання з низькою затримкою не є пріоритетом HDFS. Використовуючи HBase на HDFS ми можемо отримати низьку затримку доступу до малих файлів або таблиць з великих чанків даних [4].

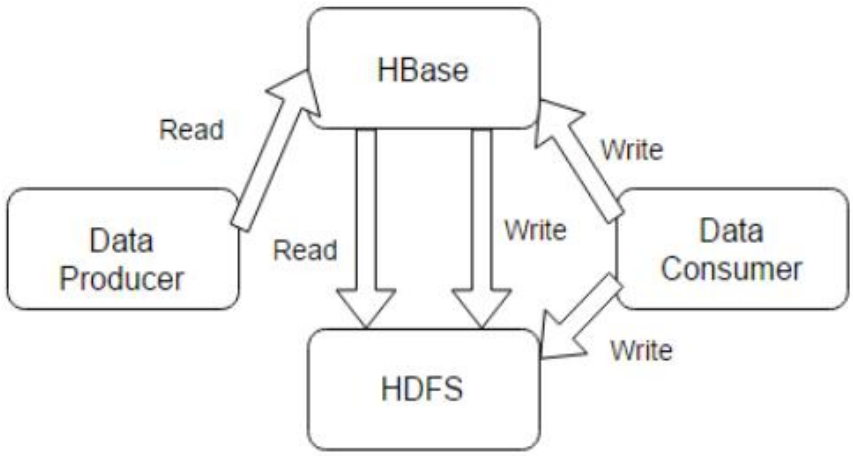

Рис. 1.6 HBase з Hadoop

Серед переваг Apache HBase можна віднести наступні пункти [11]:

- Специфічна модель даних, яка добре підходить для розріджених наборів даних.
- Аналогія з реляційними СКБД в плані індексу початкового ключа.
- Влаштований механізм часових міток, які добавляються автоматично і не можуть бути змінені вручну.

- Наявність інструментів розширеності, які дозволяються працювати з даними в HBase, як з реляційними таблицями.
- Висока швидкість роботи за рахунок кешування в пам’яті і обробка даних на стороні сервера через фільтри і підпроцесори.
- Висока доступність і відмовостійкість завдяки файлової системи Hadoop.
- Масштабованість.

Покращення часу відповіді на запит використовуючи кешування. Централізоване керування кешом є механізмом, який значно покращує час відповіді на запит файлової системи [12]. Це дозволяє користувачам встановити шлях для кешування в головній пам’яті HDFS, як показана на Рис. 1.7. В цьому випадку, NameNode комунікує з DataNodes, що мають потрібні файли на їхніх дисках і надає їм інструкції для кешування відповідних блоків в кеш-пам’яті, щоб покращити ефективність напряму надаючи кешований шлях для такого самого запиту від інших користувачів.

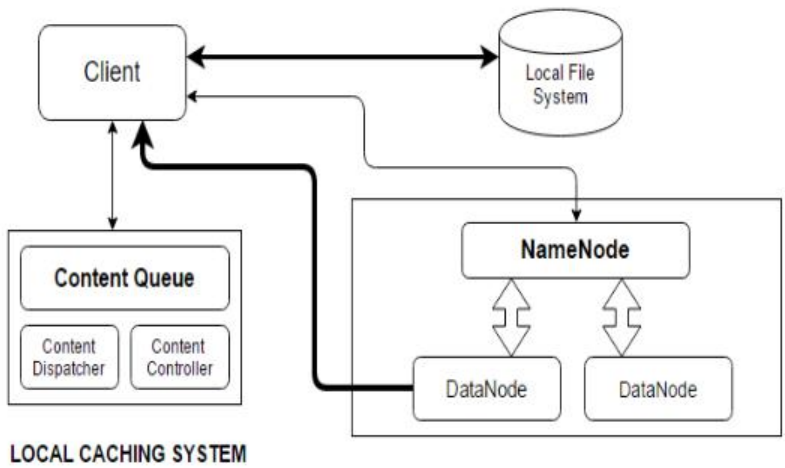

Рис. 1.7 HDFS з покращеним кешуванням

Централізоване керування кешом має багато переваг:

1. Утримання часто використовуваних даних: кешування запобігає вилученню часто використовуваних даних з пам’яті. Це корисно коли розмір робочого сету перевищує доступну основну пам’ять.

2. Ефективні операції читання: збільшення ефективності відбувається за рахунок спільного розташування завдання з кешованою реплікою блоку, керованою NameNode.

3. Використання нового API: використання нового ефективного API стає легшим, оскільки суми верифікації кешованих даних зроблені раніше на DataNodes. Отже, клієнти не отримують додаткового навантаження при використанні нового API.

Ефективне керування пам'яттю: кешування покращує використання в пам'яті в кластері в загальному. З централізованим керуванням користувач може вибрати певну кількість реплік, тим самим зберігаючи пам'ять.

1.3. Використання розподілених файлових систем

Інтерфейс файлової системи є досить загальним і підходить для широкого спектру програм, проте більшість реалізацій розподілених файлових систем оптимізовані для певного класу програм. Наприклад, Andrew File System оптимізована для користувацьких домашніх директорій, XrootD оптимізована для доступу з високою пропускною здатністю до сетів даних високо-енергетичної фізики [18], Hadoop FIle System розроблена, як шар зберігання для фреймворку MapReduce [19], CernVM File System оптимізована для розподілених бінарних файлів [20], і Lustre є оптимізованою, як простір для взаємодії програм на суперкомп'ютерах [21].

З погляду програм є різні рівні інтеграції, яку може запропонувати розподілена файлова система. Більшість файлових систем пропонують бібліотеку з інтерфейсом, що нагадує файлову систему POSIX. Перевагою, є те, що інтерфейс може бути адаптований для різних випадків. Наприклад, Google File System розширює семантику POSIX для атомного додавання, функція, яка особливо корисна для фази злиття робіт в MapReduce. Інтерфейс бібліотека реалізуються ціною прозорості. Програми повинні бути розроблені і компільовані для певної розподіленої файлової системи [14].

Системи інтерпозиції представляють шар опосередкованості, що прозоро перенаправляє виклики файлової системи програмою до бібліотеки, або зовнішнього процесу. Система Parrot створює пісочницю навколо користувацького процесу і перехоплює його системні виклики [16]. Fuse — файлова система на рівні ядра перенаправляє виклики файлової системи до спеціальних процесів користувача [22]. Системи інтерпозиції реалізуються ціною виконання для шарів опосередкованості.

Деякі розподілені файлові системи є реалізованими, як розширення ядра операційної системи (NFS, AFS, Lustre). Це забезпечує краще виконання порівняно до систем інтерпозиції, але розгортання є складним і помилки реалізації зазвичай приводять до крашу ядра операційної системи.

Розподілені файлові системи не повністю дотримуються стандарту файлової системи POSIX. Кожна розподілена файлова система повинна бути протестована з ральними програмами. Функціональність, що часто недостатньо добре підтримується в розподілених файлових системах є блокування файлів, атомне перейменування файлів і директорій, кілька жорстких посилань, і видалення відкритих файлів [14].

Висновок до першого розділу

Проаналізувавши літературні та електронній джерела можна зробити висновки про важливість розподілених файлових систем в IT-індустрії великих даних та хмарних обчислень. В останні роки збільшення попиту використання хмарних технологій показав недостатній розвиток в даній сфері, як в кількісному плані так і в технологічному. Окрім того в залежності від потреби продукту можуть використовуватися системи з різним дизайном. Відсутність універсального дизайну дає можливість запропонувати ринку нові ідеї реалізацій розподілених файлових систем. Створена у цій роботі система в майбутньому може представити новий продукт хмарних технологій на українському ринку.

РОЗДІЛ 2

Системний аналіз

Системний аналіз є процесом зібрання і інтерпретації фактів, визначення проблем і декомпозиція системи на її компоненти.

Системний аналіз використовується для вивчення системи, або її частин для визначення їх цілей. Це техніка вирішення проблем, що покращує систему і забезпечує, що всі компоненти системи працюють ефективно [24].

Система — це організований набір компонентів, або підсистем, що тісно пов’язані для досягнення спільної мети. Система має різні вхідні дані, які проходять через процеси, щоб отримати певні вихідні дані, разом це виконує бажану мету системи [25].

Система має мати три обмеження [24]:

- певна структура і поведінка розроблена для досягнення наперед визначеної цілі;
- взаємозв’язок та взаємозалежність серед компонентів систем;
- цілі організації мають вищий пріоритет, ніж цілі її підсистем.

2.1. Дерево цілей

Дерево цілей є інструментом для раціонального аналізу всіх необхідних умов і їхніх залежностей для досягнення мети. Це центральний інструмент для процесу логічного мислення.

На самому верху дерева знаходиться мета, причина, або бачення. Тобто чому ця система існує. На наступному рівні знаходиться від 3 до 5 критичних факторів успіху — головні цілі, що є необхідними для досягнення мети. Під кожним критичним фактором успіху знаходиться необхідні умови для їх виконання. Як і у випадку з критичними факторами успіху, які є необхідними для виконання Мети, необхідні умови повинні бути виконаними для виконання критичних факторів.

Дерево цілей побудоване не “логіці необхідності”, тобто такі взаємовідносини читаються, як “щоб отримати/досягти ...(вища ціль) ми мусимо отримати/досягти ... (нижча ціль) ” [26].

Після побудови дерево цілей має три функції:

1. Логічна карта майбутнього стану: оскільки мета може бути досягнутою тільки після виконання всіх необхідних умов, які очевидно ще не є досягнуті на момент побудови дерева цілей, воно дає натяк на майбутній стан.
2. Зображення теперішньої ситуації: зображення різниці між теперішнім станом і майбутнім на дереві цілей.
3. Карта розробки: з позначенням різниці і цілей, і в якому порядку, можна використовувати дерево цілей, як карту розробки.

Дана система будується з метою показати принцип роботи розподілених файлових систем.

Побудуємо дерево цілей (Рис.2.1) для визначення головного напряму дій, а також проведення системно аналізу системи розподіленого зберігання файлів.

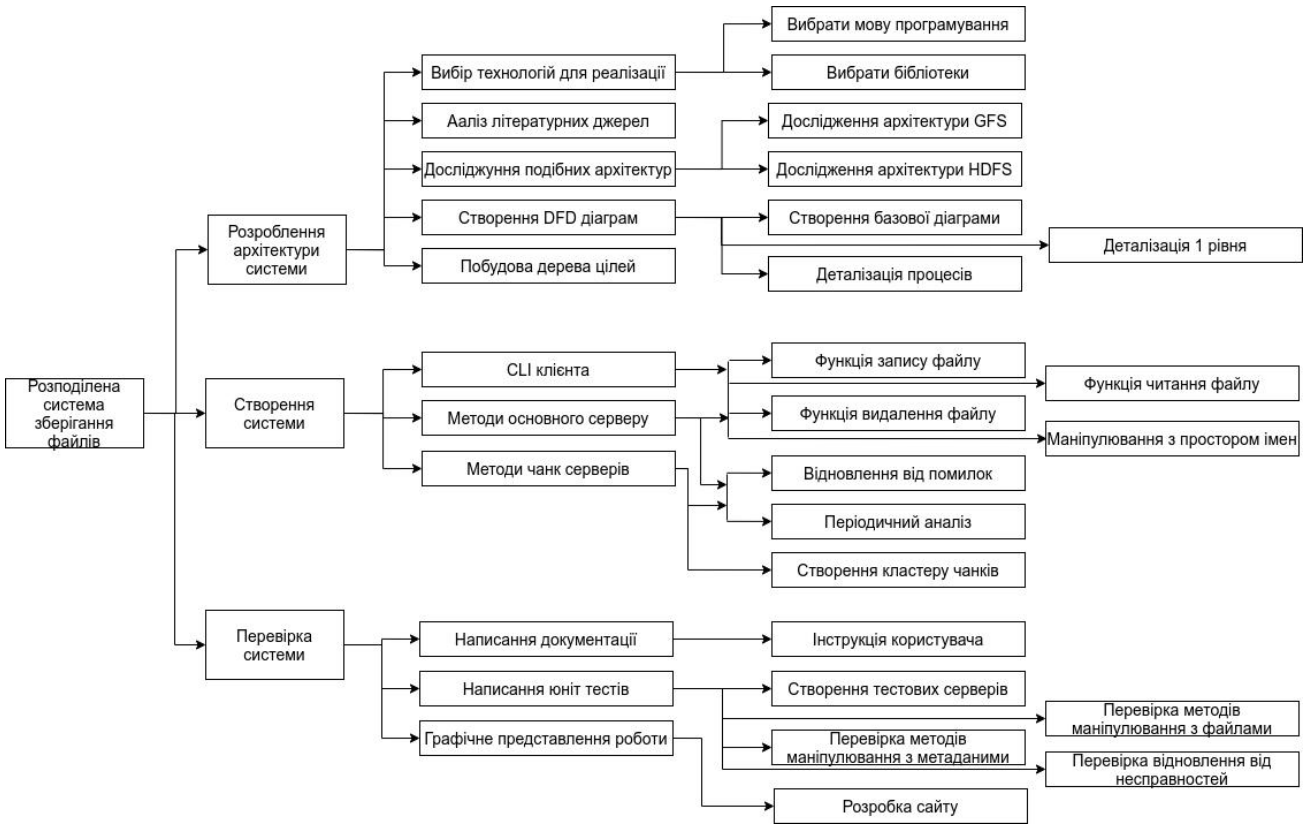

Рис.2.1 Дерево цілей

Дерево цілей має три рівні: мета існування системи, критичні фактори успіху (головні цілі), необхідні умови для їх виконання.

Метою існування системи є розроблення розподіленої файлової системи.

Для її успішної реалізації необхідне виконання трьох критичних факторів:

- Розроблення архітектури системи.
- Створення системи.
- Перевірка системи.

Розглянемо дані пункти детальніше, а також необхідні умови для їх виконання. Хоча для досягнення мети необхідне виконання всіх трьох критичних факторів, варто виконувати їх в порядку, як зазначено вище.

Необхідними умовами для виконання фактору “Розроблення архітектури системи” є:

- Вибір технологій для реалізації. Саме за допомогою них буде розроблятися дана система. Потрібно вибрати мову програмування, за допомогою якої можна буде розробити дану систему, а також бібліотеки, які знадобляться для автоматизації роботи.
- Аналіз літературних джерел. Метою якого є дослідження теми побудови розподілених систем, дослідницьких статей, книг і курсів, що допоможе в розумінні роботи архітектури, поширених проблем і помилок, а також майбутній розвиток системи. Це допоможи приймати правильні рішення при проблемах створення системи.
- Дослідження подібних систем допоможе зрозуміти популярні архітектурні рішення і їх вплив на поведінку системи. Використання досвіду при розроблені подібних систем допоможе значно скоротити час розробки. В особливості будуть досліджені такі популярні розподілені файлові системи, як GFS та HDFS.
- Створення DF-діаграм допоможе візуально продемонструвати архітектуру системи, як на абстрактному рівні, так і на більш деталізованому та продемонструвати роботу процесів.

- Побудова дерева цілей використовується для візуального представлення критичних факторів необхідних для успішної побудови системи.

Перший критичний фактор - це дослідження і аналіз теми. Для успішної реалізації системи необхідно володіти інформацією про актуальність даної теми та розуміти сучасні тренди. Також знання архітектур подібних систем значно спростить розробку.

Необхідними умовами для виконання фактору “Створення системи” є:

- CLI клієнта - це інтерфейс командного рядка за допомогою якого клієнт може викликати функції системи, а саме записати файл до системи, прочитати його та видалити. Також клієнт може маніпулювати простором імен, тобто створювати директорії, видаляти їх, виводити існуючі та змінювати робочу директорію.
- Методи основного серверу. Вони знаходяться на віддаленому серверів і викликаються RPC методами клієнта. Крім вищезазначених методів, які клієнт може викликати, він також проводить періодичний аналіз і відновлення від помилок.
- Методи чанк серверів використовуються для виконання інструкцій основного сервера та запису даних. Також для розгортання системи потрібна інструкція для розгортання цілого кластеру чанк серверів.

Другий критичний фактор описує методи необхідні для функціонування системи, як зі сторони клієнта, так і основного серверу та чанк серверів.

Необхідними умовами для виконання фактору “Перевірка системи” є:

- Написання юніт тестів допоможе переконатися в правильності роботи системи. Перевірка методів маніпулювання з метаданими, з файлами та перевірка відновлення від несправностей.
- Написання документації. В цей пункт входить інструкція користувача та загальний опис системи. Він необхідний користувачам для розуміння роботи системи та зручного початку роботи з нею. Також він необхідний розробникам для розуміння внутрішньої структури системи.

- Графічне представлення роботи відбувається за допомогою просто сайту де зображено стан системи.

Третій завершальний критерій необхідний для перевірки справності системи та представлення її роботи користувачам.

Проведемо визначення типу системи розподіленого зберігання файлів методом аналізу ієрархій. Цей структурований метод для організації і аналізу складних рішень основана на математиці і психології. Він представляє точний підхід для ваг критерій вибору. Досвід індивідуальних експертів використовується для оцінки відносної величини факторів за допомогою парних порівнянь. Кожен з респондентів порівнює відносну важливість кожної пари предметів використовуючи спеціально розроблену анкету [13].

Для початку побудуємо ієрархічну структуру задачі. А саме альтернативи і критерії, які впливають на їх вибір.

Виділимо типи альтернатив для нашої системи:

1. Надійне збереження даних (А1)
2. Зручний інтерфейс взаємодії (А2)
3. Швидкий обмін даними (А3)

Виберемо критерії для вибору типу системи:

1. Прозорість (К1)
2. Надмірність (К2)
3. Мобільність користувача (К3)
4. Простота (К4)
5. Швидкість обміну даними (К5)

Визначимо пріоритети елементів ієрархічної структури, що представляють відносну важливість або перевагу елементів побудованої ієрархічної структури, за допомогою процедури парних порівнянь. Для цього використаємо шкалу відношень подану в *Таблиці 2.1.*

Таблиця 2.1

Шкала відношень (ступеня значимості дій)

Ступінь значимості	**Визначення**	**Пояснення**
1	Однакова значимість	Дві дії мають однаковий внесок у досягнення мети
3	Слабка значимість	Існують не достатньо переконливі міркування на користь переваги однієї з дій
5	Істотна значимість	Маються надійні дані для того, щоб показати перевагу однієї з дій
7	Очевидна значимість	Переконливе свідчення на користь однієї дії перед іншою
9	Абсолютна значимість	Незаперечні переконливі свідчення на користь переваги однієї дії перед іншою
2,4,6,8	Проміжні значення між сусідніми судженнями	Ситуація, коли необхідно компромісне рішення

Побудуємо матрицю критеріїв для зображення відносної важливості критерії у порівнянні з іншими.

Таблиця 2.2

Матриця критеріїв

Критерії	**К1**	**К2**	**К3**	**К4**	**К5**	**ВЧ**	**ВВ**
К1	1,00	1,00	7,00	4,00	7,00	2,87	0,39
К2	1,00	1,00	9,00	5,00	5,00	2,95	0,40
К3	0,14	0,11	1,00	0,50	2,00	0,43	0,06
К4	0,25	0,2	2,00	1,00	2,00	0,72	0,10
К5	0,14	0,2	0,50	0,50	1,00	0,37	0,05

Розрахуємо власні числа (ВЧ) знайшовши корінь порядку 5 з добутку 5 значень критеріїв:

К1 = (1,00*1,00*7,00*4,00*7,00)^(1/5) ≈ 2,87

К2 = (1,00*1,00*9,00*5,00*5,00)^(1/5) ≈ 2,95

К3 = (0,14*0,11*1,00*0,50*2,00)^(1/5) ≈ 0,43

К4 = (0,25*0,2*2,00*1,00 *2,00)^(1/5) ≈ 0,72

К5 = (0,14*0,2*0,50*0,50 *1,00)^(1/5) ≈ 0,37

Розрахуємо власні вектори розділивши власне число критерію на суму всіх власних чисел:

К1 = 2,87/(2,87+2,95+0,43+0,72+0,37) = 0,39

К2 = 2,95/(2,87+2,95+0,43+0,72+0,37) = 0.40

К3 = 0,43/(2,87+2,95+0,43+0,72+0,37) = 0,06

К4 = 0,72/(2,87+2,95+0,43+0,72+0,37) = 0,10

К5 = 0,37/(2,87+2,95+0,43+0,72+0,37) = 0,05

Проведемо оцінювання узгодженості суджень експертів побудувавши матриці порівнянь альтернатив окремо для кожного критерію.

1. Прозорість (К1)

Таблиця 2.3

Матриця порівнянь альтернатив для прозорості

	А1	**А2**	**А3**	**ВЧ**	**ВВ**
А1	1,00	0,16	0,33	0,38	0,09
А2	6,00	1,00	5,00	3,10	0,72
А3	3,00	0,20	1,00	0,84	0,19

2. Надмірність (К2)

Таблиця 2.4

Матриця порівнянь альтернатив для надмірності

	А1	**А2**	**А3**	**ВЧ**	**ВВ**
А1	1,00	8,00	4,00	3,17	0,72
А2	0,13	1,00	0,33	0,35	0,08
А3	0,25	3,00	1,00	0,90	0,20

3. Мобільність користувача (К3)

Таблиця 2.5

Матриця порівнянь альтернатив для мобільності

	А1	**А2**	**А3**	**ВЧ**	**ВВ**
А1	1,00	0,16	0,14	0,28	0,07
А2	6,00	1,00	2,00	2,29	0,56
А3	7,00	0,50	1,00	1,52	0,37

4. Простота (К4)

Таблиця 2.6

Матриця порівнянь альтернатив для простоти

	А1	**А2**	**А3**	**ВЧ**	**ВВ**
А1	1,00	0,13	0,16	0,28	0,06
А2	8,00	1,00	5,00	3,42	0,72
А3	6,00	0,20	1,00	1,06	0,22

5. Швидкість обміну даними (К5)

Таблиця 2.7

Матриця порівнянь альтернатив для швидкості обміну

	А1	**А2**	**А3**	**ВЧ**	**ВВ**
А1	1,00	5,00	0,25	1,08	0.24
А2	0,20	1,00	0,13	0,30	0.06
А3	4,00	8,00	1,00	3,17	0.70

Побудуємо матрицю порівнянь альтернатив (*Таблиця 2.8*) за допомогою якої визначимо, яку саме альтернативу потрібно вибрати.

Таблиця 2.8

Матриця порівнянь альтернатив

Критерії	**К1**	**К2**	**К3**	**К4**	**К5**	**Узагальнені пріоритети**
	0,39	0,40	0,06	0,10	0,05	
А1	0,09	0,72	0,07	0,06	0.24	0,35
А2	0,72	0,08	0,56	0,72	0.06	0,42
А3	0,19	0,20	0,37	0,22	0.70	0,23

Після використання методу аналізу ієрархій ми отримали наступні результати:

1. Зручний інтерфейс користувача - 0,42
2. Надійне збереження даних - 0.35
3. Швидкий обмін даними - 0,23

Це показує, що найбільший пріоритет має альтернатива для зручного інтерфейсу користувача, проте пріоритети усіх трьох альтернатив є більш-менш рівними.

2.2. IDEF0 для деталізації структури

За допомогою інструменту AllFusion побудуємо IDEF0 (Рис. 2.2) для деталізації процесів системи розподіленого зберігання файлів.

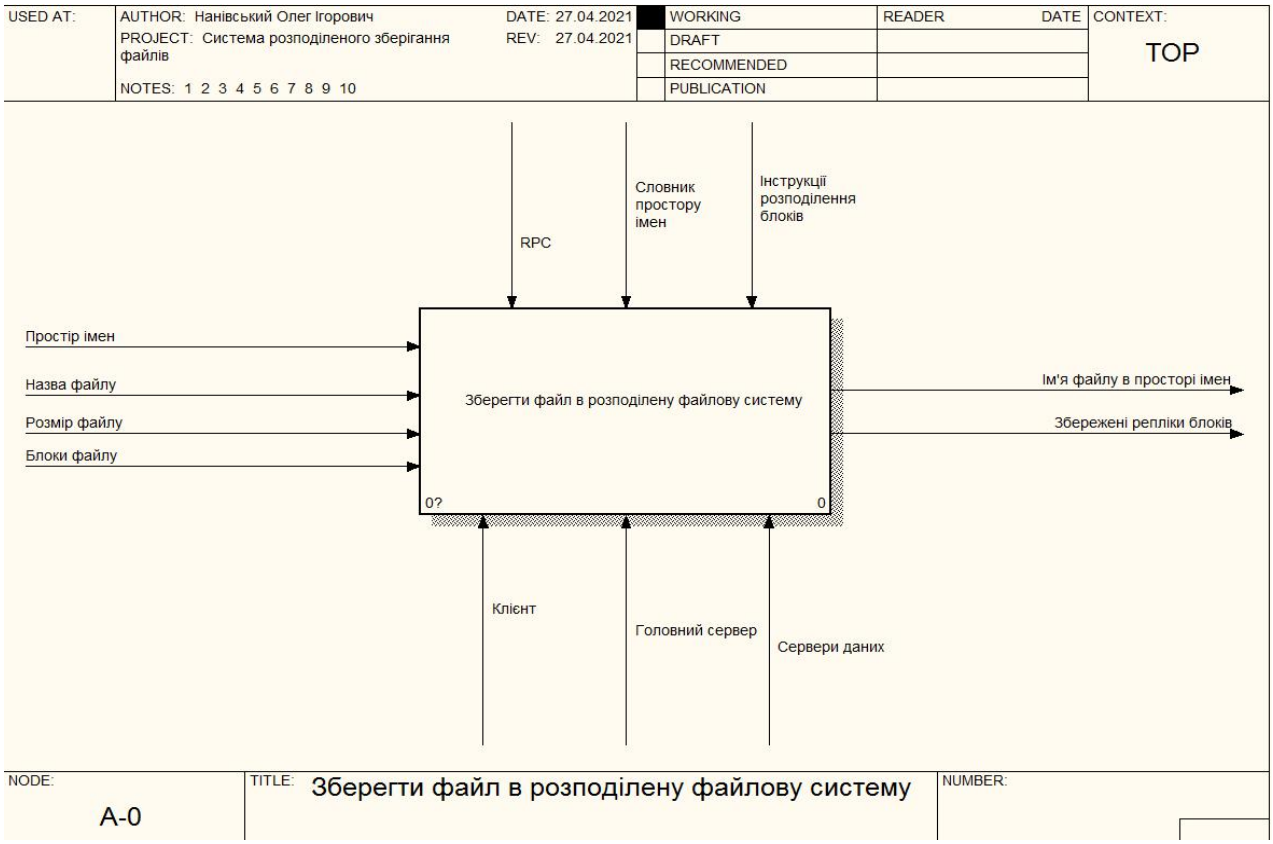

Рис.2.2 IDEF0

Процес збереження файлів в розподілену файлову систему є одним із найважливіших процесів, що може продемонструвати ефективність роботи системи.

На діаграмі IDEF0 зліва стрілки означають вхідні дані для виконання даного процесу, справа це результат його виконання. Стрілка зверху позначає, які фактори контролюють процес збереження файлу. Стрілки знизу - механізми, які залучені до виконання певного процесу.

Вхідними даними для збереження файлу є наступними:

- Простір імен - це стрічка, яка вказує на умовну директорію де буде збережена назва файлу і за якою будуть ідентифікуватися блоки файлів.
- Назва файлу.

- Розмір файлу. Він пізніше буде потрібен для визначення кількості реплік необхідних для збереження файлу цілком.
- Блоки файлу - періодичні відрізки байтів файлу, які будуть зберігатися на різних серверах даних.

Вихідними даними є:

- Ім’я файлу в просторі імен. Зареєстроване ім’я в словнику головно сервера використовується, як ідентифікатор певного файлу і його блоків.
- Збережені репліки блоків. В даній розподіленій файловій системі файл зберігається, як набір блоків і його реплік для збереження надмірності.

Факторами контрою є:

- Інструкції розподілення блоків. Вони розподіляють репліки блоків файлів рівномірно серед серверів даних.
- Словник простору імен дозволяє визначити дозволені операції із файлом.
- RPC функції, які контролюють обмін повідомленнями між клієнтом, основним сервером та серверами даних.

Механізмами залученими для збереження файлу є:

- Клієнт, який надає інформацію про файл.
- Головний сервер, що реєструє файл і надає його реплікам блоків сервери даних на яких вони будуть збережені.
- Сервери даних зберігають репліку.

Кожен з цих елементів є необхідним для успішного збереження файлу.

Наступним кроком проведемо декомпозицію контекстної діаграми (Рис. 2.3). Вона буде складатися з трьох другорядних процесів, а саме:

1. Зберегти файл. Цей процес реєструє унікальний ідентифікатор для файлу в розподіленій файловій системі використовуючи, як вхід назву файлу та простір імен. Важливо щоб не було файлу з такою ж назвою в тому самому просторі імен. Як вихідні дані ми отримаємо зареєстрований файл в словнику простору імен. Механізми, які беруть участь в процесі є клієнтом і головним

сервером, в контексті клієнт надсилає ім’я файлу, а сервер реєструє. Елементи, які контролюють даний процес є RPC виклики за допомогою яких механізми комунікують, а також словник простору імен, який вказує на можливість збереження файлу.

2. Розподілити файлу. В розподіленій файловій системі файл розбивається на блоки, і його репліки, зазвичай це три репліки для кожного блоку реєструються до відповідних серверів даних. Вхідними даним є розмір файлу, який необхідний для визначення кількості необхідних блоків для повного збереження файлу. Вихідними є даними є зареєстровані локації репліки блоків. Механізмом використаним в даному процесі є головний сервер, де і відбувається сам процес. Елементи контрою є інструкції розподілення блоків, головною метою який є розподілити блоки максимально оптимально в кластері серверів даних.

3. Записати блок. Цей процес, як вхідні дані отримує блоки файлу, які розбиває на стороні клієнту та ім’я файлу в просторі імен. Як вихід ми отримуємо збережені репліки блоків в серверах даних. Механізмами даного процесу є клієнт і сервери даних. Клієнт розбиває файл, і комунікує з сервером даних, щоб передати йому блок файлу за допомогою RPC викликів, що є одним з елементів контролю, як і локації репліки блоків, які контролюють якому саме серверу даних клієнт повинен надіслати блок.

Деталізуємо кожен процес окремо.

Спочатку розглянемо процес збереження файлу (Рис.2.4). В своїй суті це процес відповідає за перевірку доступності ідентифікатора і реєстрації його на головному сервері.

Він складається з трьох підпроцесів:

1. Надіслати назву основному серверу. Як вхід цей процес отримує назву файлу необхідну для реєстрації. Виходом є та сама назва передана головному серверу. Механізмами, які беруть участь в процесі є клієнтом який надсилає

повідомлення і головним сервером, який його приймає. Передача здійснюється за допомогою елементу контролю RPC - віддалених процедурних викликів.

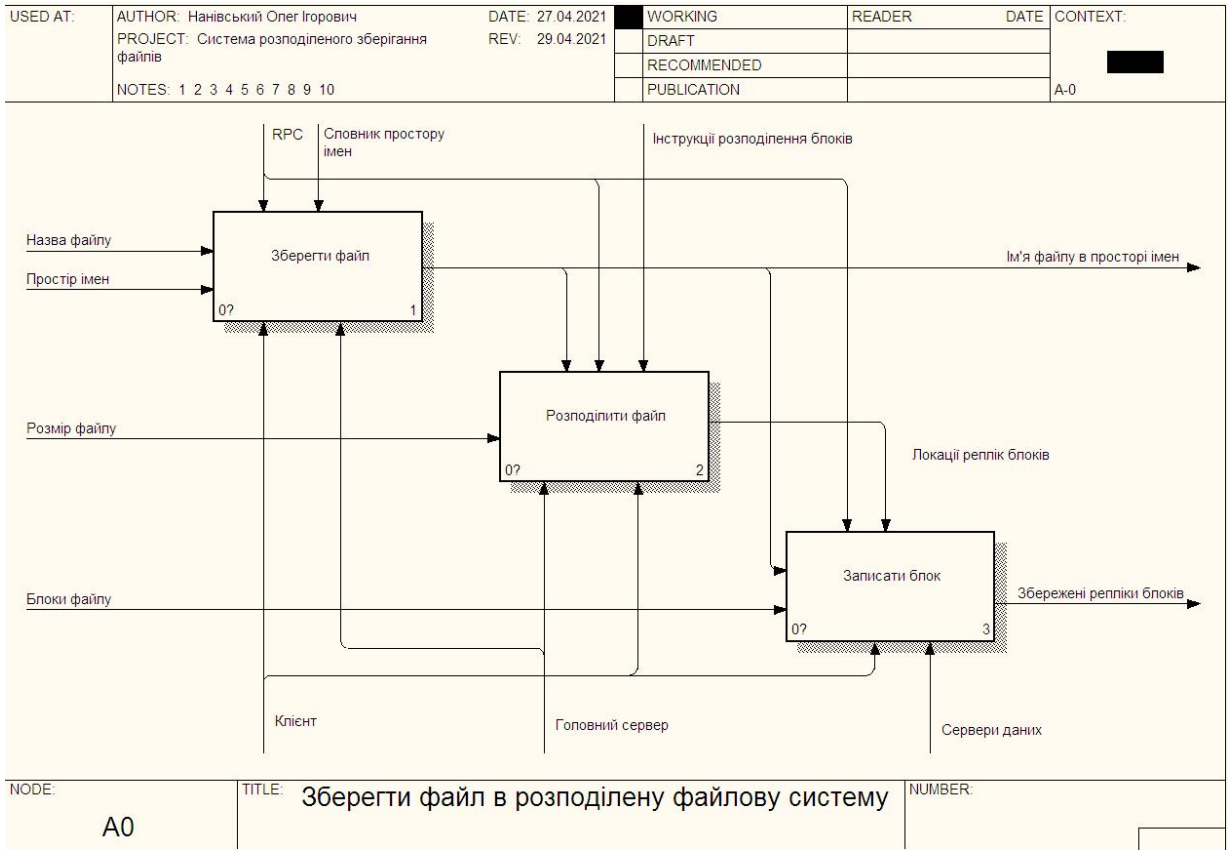

Рис.2.3 Деталізація діаграми процесів

2. Перевірити доступність простору імен. Отримавши назву файлу та простір імен куди потрібно зберегти файл на вхід основний сервер повинен спочатку перевірити чи даний ідентифікатор дозволений для запису і як результат на виході ми отримаємо можливість запису файлу в словник просторів імен на головному сервері. Головний сервер є єдиним механізмом який бере участь в даному процесі.

3. Записати простір імен в словник. В цьому процесі відбувається саме реєстрація ідентифікатора в просторі імен на головному сервері. Як вхід він отримує Простір імен в який потрібно записати файл та назву файлу передану від клієнта. Поєднання простору імен і назви файлу створює унікальний ідентифікатор. Як вихід ми отримуємо зареєстрований простір імен на головному сервері. Механізмом який бере участь в записі є головний сервер, в

той час як елементами контролю є результат попереднього процесу щодо можливості запису ідентифікатора, а також словник простору імен.

Реєстрація ідентифікатора дозволить звертатися до блоків даного файлу клієнту в майбутньому.

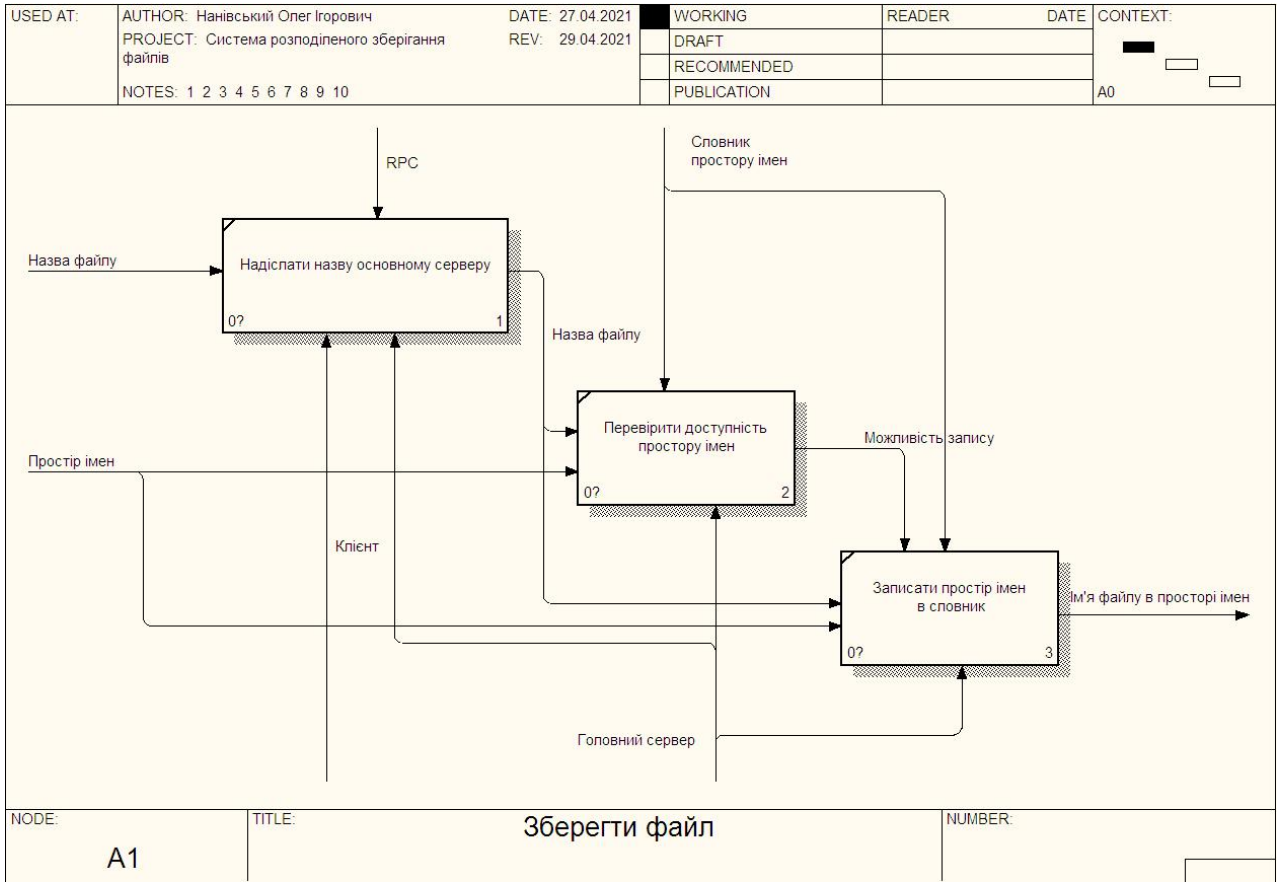

Рис.2.4 Деталізація процесу зберегти файл

Розглянемо другий процес розподілення файлу (Рис.2.5). Цей процес відповідальний за надання кожному блоку файлу місця для його збереження.

Процес розподілення файлу складається з трьох підпроцесів. Розглянемо їх детальніше:

1. Знайти кількість необхідних реплік. Цей процес необхідний, щоб знати на скількох серверах і скільки копій блоків необхідно записати. Як вхід він отримує розмір файлу від якого буде залежати кількість блоків для повного збереження файлу. Як вихід ми отримаємо кількість реплік необхідних для збереження. Механізмами які беруть участь в даному процесі є головний клієнт та головний сервер якому клієнт передає інформацію. Елементами контролю є

інструкція розподілення блоків, яка визначає яку необхідну кількість блоків необхідно для збереження файлу та RPC - віддалені процедурні виклики для передачі розміру файлу між клієнтом та сервером.

2. Знайти вільні сервери даних. Після отримання кількості необхідних реплік на вхід від попереднього процесу потрібно їх розподілити до серверів даних. Як результат на виході ми отримаємо список серверів даних на які буде записано репліки блоків в подальшому. Головний сервер є єдиним механізмом який бере участь в даному процесі, а елементом контролю є інструкції розподілення блоків. Даний процес є важливим оскільки від нього залежить ефективність роботи розподіленої файлової системи, наприклад при розподілені блоків одному серверу система втратить свою надмірність, а при розподілені до перевантажених серверів може втратити свою ефективність, саме тому розподілення повинно бути оптимальним.

3. Зареєструвати блоки серверам даних. Після знаходження серверів даних необхідно записати місце знаходження серверів даних на головному сервері. Як вхід ми отримуємо кількість реплік для запису та сервери даних на який вони повинні бути записані. Як вихід ми отримуємо зареєстровану локацію збереження репліки на сервері даних. Механізмом є лише головний сервер на якому виконується реєстрація. Елементами контролю є інструкція розподілення блоків та ім’я файлу в просторі імен, для можливості подальшої ідентифікації блоків.

Третім процесом для збереження файлу в розподілену файлову систему є запис блоків (Рис.2.6). В загальному це процес запису частини файлу на віддаленому сервері даних.

Розглянемо його підпроцеси:

1. Отримати адресу серверу даних. На вхід процес отримує локації репліки блоків даних за допомогою елементу контролю RPC, як результат попереднього процесу, а також ім’я файлу в просторі імен для ідентифікації. Механізмом є клієнт.

2. Передати блок файлу. Отримавши на вхід блоку файлу та адрес серверу даних процес за допомогою елементу контролю RPC - віддалених процедурних викликів записує частину файлу в розподіленій файловій системі. Як вихід ми отримуємо записаний блок в системі. Механізмами є клієнт який хоче записати блок та сервер даних на якому він записується.

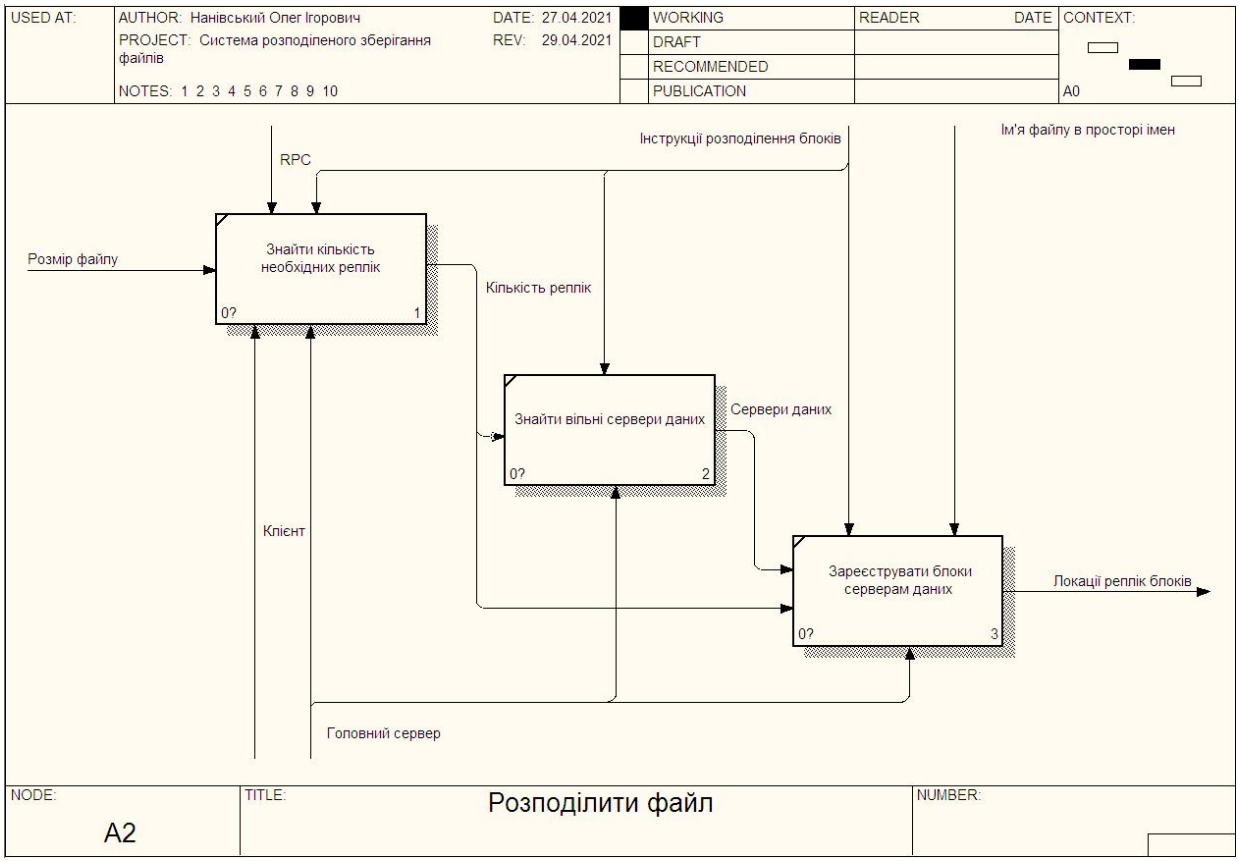

Рис.2.5 Деталізація процесу розподілити файл

Розглянувши деталізацію процесів для запису файлу в розподілену файлову систему можна спостерігати важливість і велику кількість комунікацій за допомогою віддалених процедурних викликів між клієнтом та основним сервером і в менші мірі між клієнтом і серверами даних, проте саме комунікація між клієнтом та серверами даних займає більшу частину трафіку, оскільки основний сервер маніпулює лише метаданими, а сервери даних записують великі частини файлів. Дане рішення дозволило робити швидкі, але легкі запити до системи і окремо важкі, але стабільні передачі файлів.

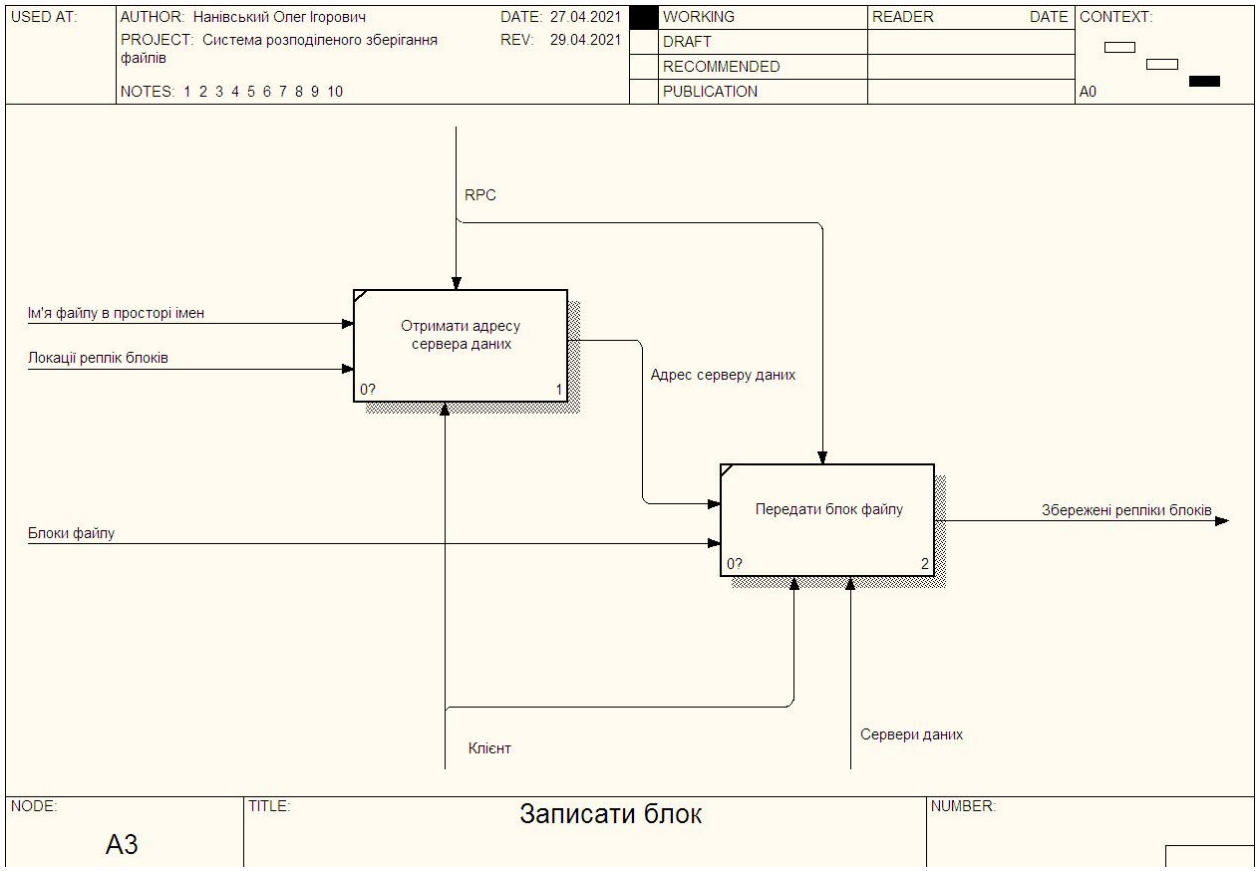

Рис.2.6 Деталізуємо процес записати блок

Серед найосновніших процесів в системі можна виділити оптимальне розподілення реплік блоків серед серверів даних та правильна реєстрація ідентифікатора на основному сервері, помилка в який може призвести до незворотних змінах в розподіленій файловій системі.

Також варто звернути на відсутність комунікації між основним сервером і серверами даних. В даному випадку це не необхідно, адже основний сервер володіє інформацією про сервери даних, як результат попередніх перевірок.

2.3. Ієрархії процесів IDEF0

На наступному етапі проведемо побудову ієрархії задач 3 рівня. Це допоможе структурувати виконувані процеси в одну місці. Ієрархія задач будується на основі вище деталізованих процесів за допомогою інструмента AllFusion Process Modeler. Ієрархія задач зображена на Рис.2.7.

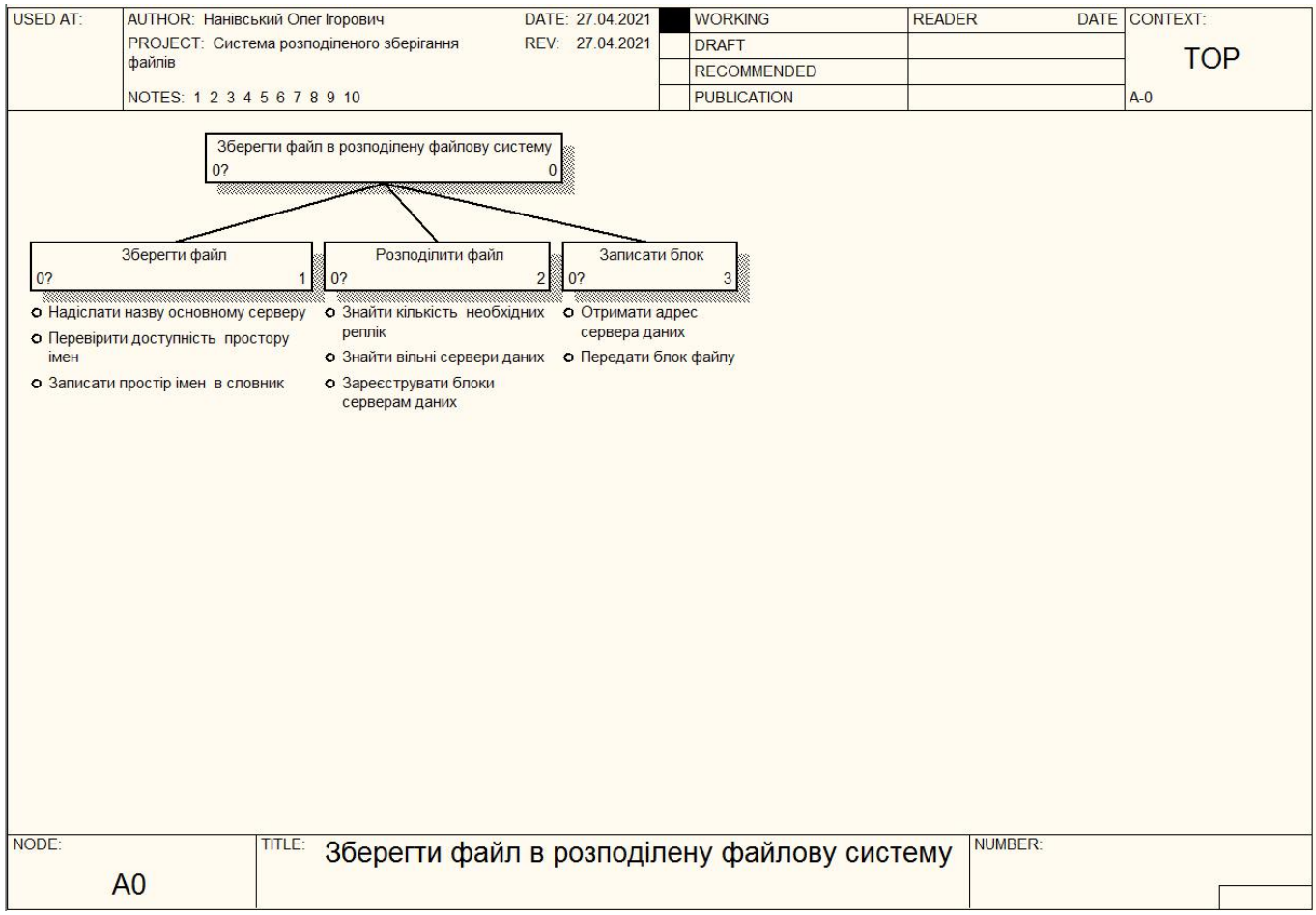

Рис. 2.7 Ієрархія задач

Основною метою “Зберегти файл в розподілену файлову систему” є одною з основних. Дана мета виконується виконанням ряду процесів: “Зберегти файл”, “Розподілити файлу”, “Записати блок”. Кожен з цих процесів складається з декількох підпроцесів:

1. “Зберегти файл”:
   a) Надіслати назву основному серверу;
   b) Перевірити доступність простору імен;
   c) Записати простір імен в словник;
2. “Розподілити файлу”:
   a) Знайти кількість необхідних реплік;
   b) Знайти вільні сервери даних;
   c) Зареєструвати блоки серверів даних;
3. “Записати блок”:

a) Отримати адрес сервера даних;

b) Передати блок файлу;

Висновок до другого розділу

Результатом виконання другого розділу стало здійснення системного аналізу розподіленої файлової системи зберігання файлів. Було побудовано дерево цілей для відстеження необхідних умов для побудови системи. Проаналізовано можливі альтернативи побудови за допомогою метода аналізу ієрархій. За допомогою інструментів AllFusion Process Modeler було побудовано контекстну діаграму для одного з основних цілей розподіленої файлової системи, а саме збереження файлу, а також проведення її декомпозиція методом idef0 і деталізація їх процесів.

РОЗДІЛ 3

Опис програмних засобів

3.1. Обґрунтування вибору засобів розв'язання задачі

Після проведеного аналізу альтернатив, архітектур подібних систем та задач необхідних для створення даного продукту, було зроблено висновки, що для її розробки необхідна мова програмування загального призначення.

Для розробки цієї системи було вибрано мову програмування Python - це інтерпретована об'єктно-орієнтована мова програмування високо рівня. Вона була розроблена Гвідо ван Россумом в 1990р. Її високорівневі структури даних об'єднані з динамічним типізацією і динамічним пов'язанням роблять її привабливою для швидкої розробки програм, а також, як використання скриптової мови для комунікації між уже існуючими компонентами. Python проста, легка для навчання і читання мова, що значно зменшує вартість підтримки програм. Python підтримує модулі і пакети, які дозволяють модульність програм і перевикористання коду. Інтерпритатор Python і широка стандартна бібліотека доступні в джерелі для більшості популярних платформ, і може бути вільно поширеною.

Python популярна тим, що вона надає збільшення продуктивності. Оскільки вона немає етапу компіляції, цикл розробки редагування і відлагодження є надзвичайно швидким. Відлагодження Python програм є простим, оскільки помилка, або поганий ввід ніколи не призведе до помилки сегменту, а підніме виняток при знаходженні помилки. Коли програма не знаходить помилку, інтерпритатор виводить відстеження стеку. Відлагодження на рівні джерела дозволяє перевірити локальних і глобальних змінних, оцінити винятки, встановити точки зупинки, виконувати лише рядок коду за раз [15].

Для розробки даного програмного забезпечення було вибрано версію Python3.7. Це відносно нова версія з широким списком підтримуваних бібліотек, що значно спростить розробку системи.

Порівняно з іншими мовами програмування Python є досить популярним, що робить розробку з ним простішою в зв’язку з великою підтримкою від ком’юніті (Рис.3.1). Проведемо порівняльний аналіз з іншими мовами програмування [23].

- Як правило, програми Python працюють дещо повільніше, ніж програми Java, але на їх розробку також потрібно набагато менше часу. Програми Python, як правило, в 3-5 разів коротші по кількості рядків коду, ніж еквівалентні програми Java. Цю різницю можна віднести до вбудованих у Python типів даних високого рівня та його динамічної типізації.
- Python підтримує подібний стиль програмування як і JavaScript, що використовує прості функції і змінні без визначення класів. Проте на відмінну від останньої, Python також підтримує написання набагато більших програм, і краще використання коду завдяки об’єктно-орієнтованому стилю програмування, де класи відіграють важливу роль.
- Python та Perl мають багато подібних особливостей, проте мають іншу філософію. Perl наголошує на підтримці загальних завдань, орієнтованих на додатки, наприклад завдяки вбудованим регулярним виразам проведення сканування файлів та створення звітів. Python, в свою чергу, наголошує на підтримці загальноприйнятих методологій програмування, таких як дизайн оснований на структурах даних та об'єктно-орієнтоване програмування. Як наслідок, Python подібний до Perl, але рідко перевершує його у його спеціалізованих завданнях, однак Python застосовується далеко за межами ніші Perl.
- Порівнюючи Python з C++ можна прийти до тих ж висновків, що і порівняння з Java. Python набагато повільніший за C++, проте в свою чергу набагато гнучкіший в розробці і підтримці.

Порівнюючи Python з іншими мовами програмування можна побачити, що він дещо програє Java та C++ в швидкості роботи, проте компенсує це швидкістю розробки, широкому вибору бібліотек, легкому відлагодження та легкій підтримці.

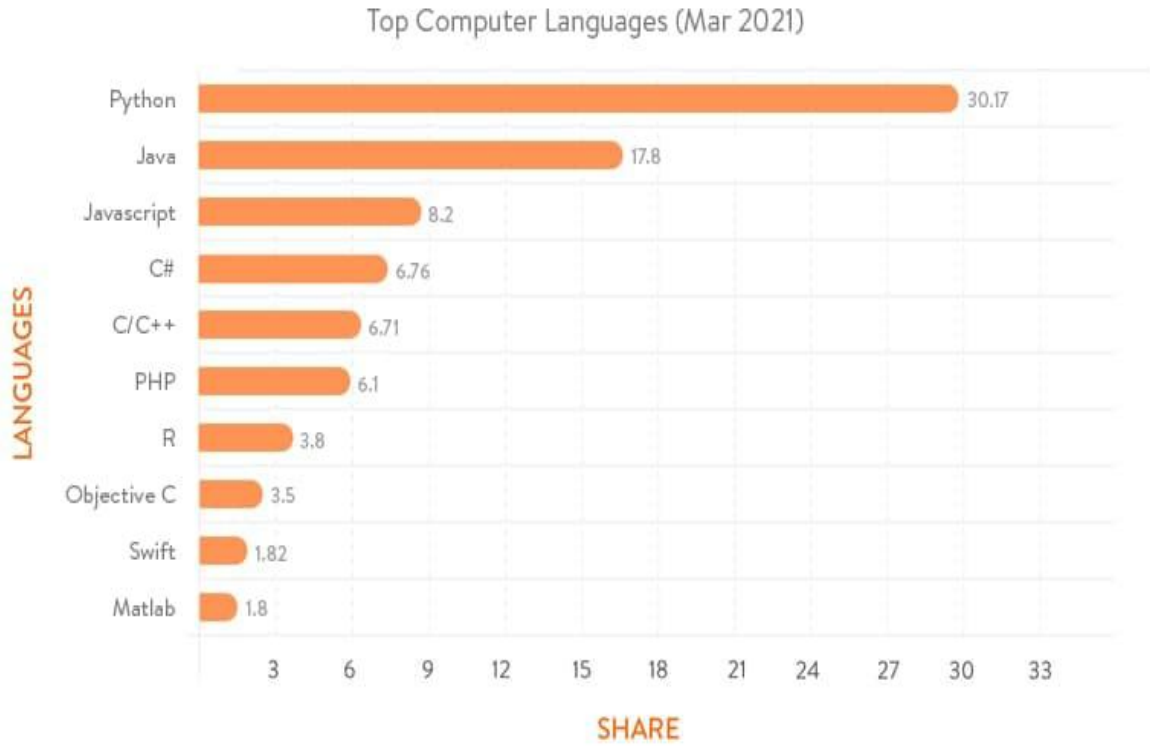

Рис.3.1 Популярність мов програмування

Враховуючи простоту розробки програм використовуючи Python, а також архітектуру системи, яка в більшій мірі основана на комунікації між серверами дана мова програмування цілком підходить для розробки системи розподіленого зберігання файлів.

При виборі мови програмування нам, також потрібно звернути на наявність потрібних бібліотек для розробки системи, а саме бібліотек для віддалених процедурних викликів.

Для початку розглянемо, що таке віддалені процедурні виклики.

Віддалені процедурні виклики (Remote Procedure Call - RPC) є методом комунікації між процесами, що використовується для клієнт-серверних додатків. Вони також відомі, як виклики підпрограм, або виклики функцій.

Клієнт має запит, що RPC перекладає і надсилає до сервера. Цей запит може бути викликом процедури, або функції на віддаленому сервері. Коли сервер отримує запит, він надсилає необхідну відповідь назад до клієнта. Клієнт є

заблокованим допоки сервер обробляє виклик і відновлює роботу тільки коли сервер закінчив [27].

Послідовність подій в віддаленому процедурному виклику:

1. Клієнт викликає функцію віддаленого виклику.
2. Ця функія викликає системний виклик, щоб надіслати повідомлення до серверу і надати параметри повідомленню.
3. Повідомлення надсилається з клієнта до сервера за допомогою операційної системи клієнта.
4. Операційна система серверу передає повідомлення до функції віддалених викликів.
5. Параметри отримуються з повідомлення за допомогою серверної функції віддалених викликів.
6. Дана викликає потрібно процедуру на сервері.

Даний процес зображено на рисунку Рис.3.2.

Перевагами віддалених процедурних викликів є:

- Віддалені процедурні виклики підтримують моделі орієнтовані на процеси та потоки.
- Внутрішній механізм передачі повідомлень прихований від користувача.
- Вимагає мінімальних зусиль для модифікації коду.
- Віддалені процедурні виклики можуть бути використані в розподілених системах та локальних середовищах.
- Багато шарів протоколів не використовуються для збільшення продуктивності.

Недоліки віддалених процедурних викликів є:

- Не існує єдиного стандарту для створення RPC.
- Не має гнучкості для апаратної архітектури. Вони основані тільки на взаємодії.
- Можливе збільшення в вартості через віддалені процедурні виклики.

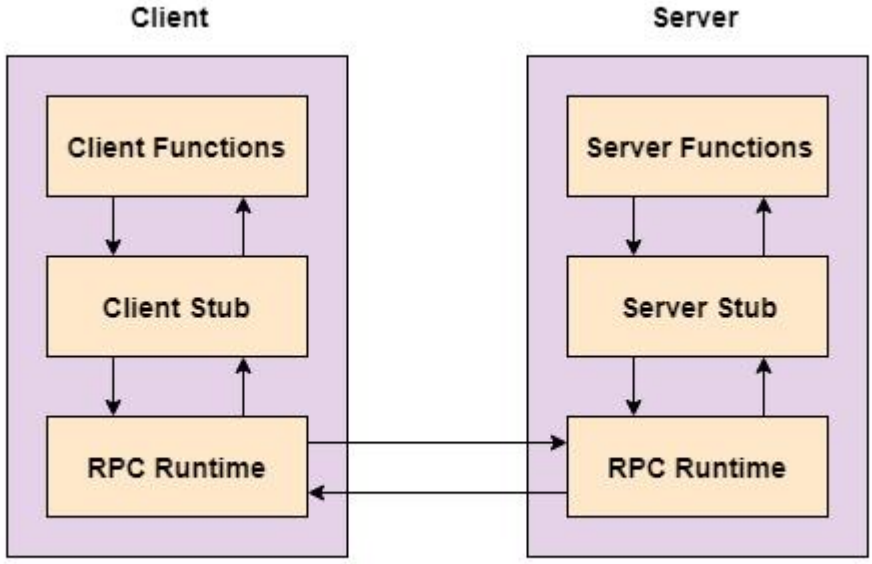

Рис.3.2 Процес віддаленого виклику процедури

Для виконання віддалених процедурних викликів виберемо бібліотеку RPyC - прозору python бібліотеку для симетричних віддалених процедурних викликів, створення кластерів і розподіленого обчислення. RPyC використовує об'єктне проксі - метод, що використовує динамічну особливість python, щоб оминути фізичні обмеження між процесами та комп'ютерами для маніпулювання віддаленими об'єктами наче вони є локальними [28].

RPyC часто використовують в наступних випадках:

- Відмінний для середовищ тестування - запускає тести з центральної машини пропонуючи зручне середовище розробки в той час як операції виконуються на віддаленій машині.
- Контролювати кілька апаратних чи програмних платформ з одної центральної точки можливо, для будь-якої машини, що підтримує Python.
- Отримання доступу до апаратних ресурсів прозоро.
- Розширення функціоналу локального, або віддаленого коду.
- Розподілення навантаження між багатьма машинами.
- Імплементація віддалених сервісів.

Взявши до уваги зручність обміну інформацією між віддаленими машинами, бібліотека RPyC цілком підходить для виконання поставленої задачі.

Для легкої розробки даної системи необхідно вибрати текстовий редактор, який би задовільняв ряд вимог: легкий, крос-платформний, підтримка мови програмування Python, інтеграції із github та зручність зневадження, безплатний. Врахувавши вище наведені фактори було вибрано VSCode (Рис.3.3).

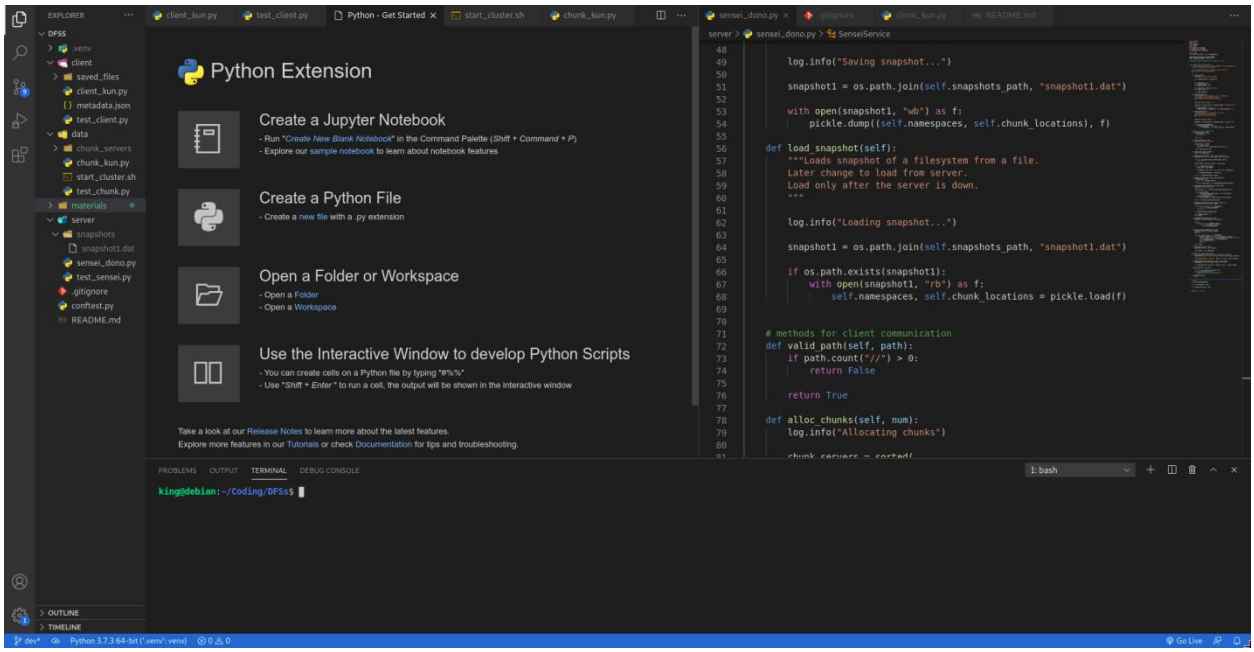

Рис.3.3 Інтерфейс VSCode

Visual Studio Code - це редактор коду на основі Electorn і розробляється компанією Mivrosoft. Його ряд переваг робить одним з кращих текстових редакторів на рику:

1. Відкрите джерело. Цей факт робить програму не тільки безплатною, а й надає можливість допомогти в її розробці, що приваблює досвідчених розробників для покращення продукту.

2. Простота. Від встановлення програми до додавання нових розширення відбувається просто та швидко. Завдяки своїй розширювальній архітектурі текстовий редактор може бути простим, але водночас мати функціонал, як в IDEs. Також простота виявляється в його продуктивності, яка досить непогана.

3. Мінімальний дизайн. Базова версія продукту надає мінімальний, але елегантний дизайн, проте є також можливість створення власного, або вибір уже готового.

4. Розширення. VSCode надає тисячі розширень для будь-яких програмних рішень. Також є можливість створення власних розширення і додавання їх до магазину розширень. Це надає розробникам можливість підлаштувати програму для власних вимог.

Для розробки розподілених файлових систем необхідна платформа для її роботи. В комерційному світі більшість серверів використовують Linux, як платформу для роботи, це більше ніж 90% всіх хмарних архітектур [32]. Проте також є підтримка і на користь Winodows і її захищеність і знайомість, однак більшість сисадмінів підтримують Linux і її відкритість. На відміну від Windows вона не обмежує розробників і надає більшу гнучкість при розробці. Linux пропонує вихід для цієї закритої екосистеми, дозволяючи компаніям використовувати декількох постачальників для надання своїх послуг. Однак, організації, які переходять у середовище, яке працює лише для Linux, цілком можуть знайти сервіси, на які вони колись покладались, не підтримуються Linux.

Однією з найбільших переваг Linux над Windows є те, що Linux є безплатною для встановлення. Проте для безпечного запуску серверів все таки знадобляться додаткові послуги, які не є безплатними, проте в більшості випадків дешевші ніж в Windows Server.

Linux є більш стабільною системою для запуску серверів, ніж його конкурент, із нижчим рівнем помилок і відмов. Linux також може підтримувати більше одночасних процесів і не повинен перезавантажуватися настільки часто. Це є наслідком того факту, що він є високоефективним через те, що він має досить небагато речей, які неабсолютно точно повинні бути тут. Тому тут є менше можливостей, щоб щось пішло не так і спричинило хаос в системі.

Вважається, що Linux є досить легким на ресурси. Більшість дистрибутивів можуть запускатися на досить старих, або слабких апаратних забезпеченнях,

наприклад Raspberry Pi. Це означає, що досить легко запустити сервери без додаткових обладнань. Хоча це не є проблемою для великих компаній, це може допомогти для нових компаній.

Іншою особливістю Linux і інших відкритих систем є переконання в те, що вони набагато надійніші від інших. Оскільки кожен може переглянути код і знайти помилки та допомогти з виправленнями.

3.2. Технічні характеристики обраних програмних засобів

Технічні характеристики обраних засобів в значній мірі впливають на процес розробки системи та її роботу, що вимагає детальнішого аналізу.

Розглянемо технічні характеристики мови Python. Python є динамічною, високорівневою, безплатною і інтерпритованою мовою програмування. Вона підтримує об’єктно-орієнтоване програмування та процедурно орієнтоване [29].

Особливості Python:

1. Легке написання коду. Python є високорівневою мовою програмування. Python є досить легким для вивчення порівняно з мовами, такими, як C, C#, JavaScript, Java. Досить легко писати код використовуючи Python і будь-хто може вивчити основи за кілька годин, або днів.
2. Безплатна і з відкритим кодом. Мова Python вільнодоступна на офіційному сайт. Оскільки вона з відкритим кодом, це означає, що код доступний для публіки. Отже, є можливість завантажити її, а також поділитися.
3. Об’єктно-орієнтована. Однією з особливостей мови є її об’єктно-орієнтоване програмування. Python підтримує концепти класів, інкапсуляції і інші.
4. Підтримка розробки інтерфейсів. Графічні інтерфейси можуть бути розроблені використовуючи модулі такі, як: PyQt5, wxPython, Tk.
5. Високорівнева мова. Python є високорівневою мовою. При написанні програм на python нам не потрібно знати архітектуру системи, або взаємодіяти із пам’яттю.

6. Можливість розширення. Python дає можливість записати деякий python-код або компілювати його в C/C++.

7. Переносна мова. Python програми можна запускати на різних платформах, таких як, Linux, Windows, Mac, Unix.

8. Інтегрована мова. Python надає можливість інтеграції із іншими мовами програмування.

9. Інтерпритована мова. Програми виконуються один рядок за раз, що дозволяє простіше відлагодити код. В внутрішній структурі Python-код перетворюється в bytecode.

10. Велика стандартна бібліотека. Існує велика стандартна бібліотека, яка надає модулі для великої кількості загальних задач, що дозволяє не писати код один і той самий код кожен раз.

11. Динамічно типізована мова. Це означає, що тип змінних визначається в процесі виконання програми.

Розглянувши особливості мови програмування Python варто також розглянути і її обмеження. Розглянемо найбільш важливі обмеження:

1. Продуктивність та швидкість. Численні дослідження показали, що Python в загальному повільніша, ніж інші сучасні мови програмування такі, як Java, C++. Проте існує декілька можливостей для програм працювати швидше. Однією з них є написання власного часу виконання програми і використовувати його замість стандартного.

2. Несумісність двох версій. Офіційно друга версія Python є застарілою, але обидві версії, друга і третя, оновлюються на регулярній основі. Також досі велика кількість програмістів надають перевагу другій версії враховуючи велику кількість популярних фреймворків та бібліотек, які підтримують тільки другу версію.

3. Переносимість додатків. Python є високорівневою мовою програмування. Отже розробники використовують інтерпритатори для перетворення коду зрозумілого для операційної системи. Тому необхідне

встановлення певної версії інтерпритатора на операційну систему, щоб запустити програму на даній платформі.

4. Вимагає додаткового тестування. В зв'язку з динамічною типізацією в процесі написання програми не потрібно явно позначати тип змінною, що спрощує написання програми, проте це може привести до помилок виконання. Тому розробникам потрібно проводити тестування для визначення можливих помилок.

5. Залежить від сторонніх фреймворків та бібліотек. Python не має функціоналу, який є у інших сучасних мовах програмування. Отже, розробники використовують сторонні інструменти, що робить майбутній продукт залежним від даних інструментів.

6. Деякі модулі не мають адекватної підтримки. Python підтримується великим ком'юніті, яке часто модифікує нові модулі для збільшення їхньої функціональності, проте не всі модулі оновлюються часто, або мають однакову якість. Це вимагає від розробників дослідити, який модуль їх підходить.

В загальному розробка Python програм повинна підтримуватися рядом сторонніх бібліотек, або фреймворків для подолання обмежень мови [30].

Бібліотекою для комунікації між елементами системи є RPyC. Дана бібліотека володіє наступними особливостями:

1. Прозорість - доступ до віддалених об'єктів, як до локальних; існуючи код працює правильно як з локальними так і з віддаленими об'єктами.

2. Симетричність - протокол є повністю симетричним, тому і клієнт і сервер можуть обслуговувати запити.

3. Підтримка синхронних і асинхронних операцій.

4. Агностичний до платформ - windows/linux/solaris/mac і інші.

5. Низькі накладні витрати. Бібліотека використовує компактний двійковий протокол і не вимагає складного встановлення.

6. Захищений - легко інтегрується із SSH.

7. Володіє функцією нульового розгортання.

8. Інтегрується із TLS/SSL, SSH і inetd.

Для ефективної роботи із даною бібліотекою необхідне знання її внутрішніх механізмів.

Один із найбільш фундаментальних концептів в комп'ютерному програмуванні є процес. Процес - це одиниця коду і даних розміщеному в унікальному адресному просторі, що забезпечує ізольованість його від інших процесів. Це дозволяє запускати кілька процесів на одному апаратному забезпеченні без впливу один на одного. Із перспективи RPyC процеси накладають штучні обмеження між програмами, що змушує програми до монолітної структури.

Одним із механізмів для подолання цього є віддалені процедурні виклики (RPC). RPC дозволяє виконувати код процесу, який знаходиться в іншому адресному просторі і дізнаватися результат виконання цього коду. Є багато RPC фреймворків, які діляться такими базовими ознаками: опис, які функції відкриті для виклику, встановлюють формат серіалізації, передачі абстракції і створенні коду на стороні клієнту для можливості виклику віддалених функцій.

На відміну від більшості RPC, RPyC є прозорою, що дозволяє додавати її до існуючого коду практично без додаткових витрат. Як наслідок прозорості бібліотека є також симетричною. Це дозволяє не поділяти програми на клієнта і сервер, оскільки обидві програми можуть подавати запити і відповіді.

Прозорість і симетричність дозволяє двом об'єктам локальному і віддаленому бути однаковими перед кодом. Іншими словами два процеси об'єднанні за допомогою RPyC можуть розглядатися як один процес [31].

Розглянемо технічні характеристики вибраного редактору коду Visual Studio Code. Нижче перелічено його особливості, деякі з них є унікальними для нього:

1. Підтримка багатьох програмних мов. Раніше програмісти використовували різні текстові редактори для різних мов, проте у VSCode є можливість використовувати практичну будь-яку популярну мову програмування. Це також дозволило легко визначати помилки, або посилання між мовами.

2. Intelli-Sense. Це дозволяє визначати, які частини коду залишилися не заповненими. Також звичайні декларація змінної і її синтаксис є автоматичними. Тобто якщо якась змінна використовується в коді, але розробник забув її декларувати то редактор зробить це автоматично.

3. Підтримка різних платформ. VSCode може працювати на платформах Windows, Linux та Mac.

4. Розширення та підтримка. Редактор в підтримує велику кількість мов, проте, якщо деякі з мов не підтримуються розробник може встановити розширення, яке її підтримує. Також розширення не уповільнюють програму, адже вони працюють, як окремі процеси.

5. Репозитарій. Редактор з'єднаний із Git для безпечного збереження коду.

6. Підтримка Web. Web додатки можуть бути побудовані і підтримувані в VSCode редакторі.

7. Ієрархічна структура. Файли коду знаходяться в файлах і директоріях. Необхідні файли коду також мають файлу, які можуть бути потрібні для інших складних проектів.

8. Покращення коду. Деякі частини коду можуть бути декларовані дещо інакше, що може допомогти користувачу в написанні коду. Ця функція може пропонувати користувача написати певну частину коду в інший спосіб.

9. Підтримка терміналу. В багатьох випадках користувачу потрібно почати із кореневої директорії для певної дії. Вбудований термінал, або консоль забезпечує користувачу можливість не переключатися між вікнами.

10. Багато проектів. Можливість відкриття директорій і їх файлів для багатьох проектів одночасно.

11. Підтримка Git. Редактор має вбудовану підтримка для базового git функціоналу, таких як клонування коду із репозитарію github, і його оновлення та інших.

12. Коментування. Популярна особливість, але не всі мови програмування підтримують її. Вона дозволяє пригадати функціональність написаного коду.

Як платформу для роботи розподіленої файлової системи було вибрано операційну систему Linux. Від неї залежить ефективність роботи системи тому знання з ефективної її встановлення та підтримки є необхідними.

Linux Server - це динамічна модифікація операційної системи Linux, розроблена і створена Лінусом Торвальдсом в 1991 році. Він спеціально розроблений для управління більш вимогливими потребами бізнес-додатків, таких як адміністрування мережі та системи, управління базами даних, веб-сервіси та інших.

Деякі з провідних ОС Linux Server включають Debian, Ubuntu, Slackware. Вони мають кілька різноманітних функцій, які відрізняють їх від інших операційних систем, але для початку розглянемо його архітектуру.

Архітектура ОС Linux має наступні п’ять компонентів:

1. Ядро - основна частина будь-якої операційної системи Linux. Це комп'ютерна програма яка відповідає за всі основні дії ОС. Вона складається з модулів пам'яті, які взаємодіють безпосередньо з апаратним забезпеченням. Ядро володіє абстракцією для приховування прикладних програм, або деталей апаратного забезпечення для системи. Можуть використовуються різні типи ядер, такі як монолітні ядра, екзоядра, мікроядра тощо.
2. Системна бібліотека - це набір заздалегідь визначених програм, або спеціальних функцій, за допомогою яких прикладні програми (або системні утиліти) отримують доступ до функцій ядра. Як правило, вони реалізують функціональні можливості операційної системи, а отже, не потребують коду доступу з модуля ядра.
3. Системні утиліти - це прикладні програми, які відповідають за спеціалізовані та окремі завдання за вказівкою ядра та системних бібліотек.
4. Апаратне забезпечення - як правило, воно складається з оперативної пам'яті, жорсткого диска, процесора тощо.
5. Оболонка - діє як інтерфейс між користувачем та ядром; вона приймає команди від користувача і передає їх ядру для виконання. Оболонки, присутні в

різних ОС, класифікуються на дві категорії: командний рядок та графічна оболонка. Як випливає з назв, оболонка командного рядка забезпечить інтерфейс командного рядка, тоді як графічна оболонка надає графічний інтерфейс користувачеві. Останній повільніший, хоча обидва вони здатні виконувати однакові дії.

Розглянемо деякі особливості ОС Linux:

- Індивідуальні клавіатури:

Завдяки своїй доступності кількома мовами, Linux має багато користувачів по всьому світу. Ця функція дозволяє настроювати клавіатуру відповідно до мовних потреб людей з різних країн.

- Не вимагає встановлення:

Майже всі розподілені послуги і системи Linux постачаються із включеною функцією Live CD або USB. Це дозволяє клієнтові використовувати та запускати операційну систему без необхідності установки.

- Підтримка додатків:

ОС або Linux Sever має арсенал програмного забезпечення або сховище, куди можна завантажити велику кількість програмного забезпечення через термінал або оболонку Linux. Він може навіть запускати програмне забезпечення Windows.

- Графічний інтерфейс користувача:

Незважаючи на те, що Linux має інтерфейс командного рядка, він має функції графічного встановлення системи інтерфейсу користувача на базі Windows.

- Портативність та відкритий код:

ОС Linux також може використовуватися на різних типах обладнання; більше того, ядро Linux підтримує встановлення будь-якої апаратної платформи. Linux має операційну систему з відкритим кодом, що означає, що вихідний код цієї операційної системи є у вільному доступі для всіх, для використання та модифікації, що проводиться різними групами, для покращення її можливостей та продуктивності в майбутньому.

- Використання багатьох користувачів та мультипрограмування:

Кілька користувачів можуть одночасно отримати доступ до системних ресурсів ОС Linux, таких як оперативна пам'ять, програми тощо, що робить її багатокористувацькою ОС. Крім того, одночасно можна запускати кілька додатків, що робить можливе мультипрограмування. Це дуже практична особливість операційної системи Linux.

- Ієрархічна файлова система:

ОС Linux не тільки складається зі стандартної файлової структури для впорядкування системних файлів та користувацьких файлів, але також відома як система ієрархії файлів або FHS. Тут розташування файлів показано у кореневій директорії, незалежно від пристроїв їх зберігання; віртуальний або фізичний. Деякі з директорій:

1. / root: - коренева або основна директорія для всієї файлової системи.
2. / bin: - двійкові файли команд користувача.
3. / boot: - файли завантажувача для ядра та інші.
4. / dev: - необхідні файли пристрою.
5. / etc: - унікальні для хоста загальносистемні файли пристроїв, що містять файли конфігурації, необхідні для всіх програм.
6. / home: - файли домашнього каталогу користувачів, що містять особисті налаштування.
7. / lib: - бібліотеки, необхідні для двійкових файлів у / bin та / sbin.
8. / media: - точки кріплення для знімних носіїв, таких як CD-ROM тощо.
9. / mnt: - тимчасово змонтовані файлові системи.
10. / opt: - додаткові пакети прикладного програмного забезпечення.
11. / sbin: - необхідні двійкові файли системи.
12. / srv: - специфічні для сайту дані; дані та сценарії для веб-серверів тощо.
13. / tmp: - тимчасові файли, які можуть мати обмеження в розмірі.

14. / usr: - вторинна ієрархія даних користувачів, доступних лише для читання.

15. / proc: - віртуальна файлова система, що надає інформацію про процес та ядро у вигляді файлів.

Ієрархічна структура також дуже корисна для стандартного розташування файлів.

- Хороша безпека:

Linux надає пріоритет захисту конфіденційної інформації користувачів від потенційних загроз, таких як хакери або несанкціонований доступ до системи. Ця особливість також є причиною популярності ОС Linux на світовому ринку. Система Linux використовує такі функції безпеки, як автентифікація, авторизація та шифрування. Таким чином, очевидно, що система Linux безпечна у використанні. Таким чином, можна також зробити висновок, що ОС Linux є не лише корисною та практичною, простою у використанні з дуже зручним для користувача інтерфейсом, але також безпечною та надійною операційною системою з низкою серверів по всьому світу [33].

Висновок до третього розділу

Для побудови даної системи було вирішено вибрати мову загального призначення Python завдяки її зручності розробки і наявності необхідних бібліотек для спрощення розробки. Серед вибраних бібліотек є RPyC для віддалених процедурних викликів та pytest для тестування правильності виконання програми. Дана система розроблялася під систему Linux враховуючи, що більшість серверів використовують цю операційну систему. Текстовим редактором було вибрано VSCode.

РОЗДІЛ 4

Практична реалізація

4.1. Опис програми

4.1.1. Загальні відомості про програму

Дана розподілена файлова система складається з 3 частин: клієнтського інтерфейсу командного рядка, основного серверу для роботи з метаданими та сервера даних для збереження даних. Кожен з даних елементів є незамінним в правильному функціонуванні системи. Клієнтська частина програми надається клієнту, як інтерфейс командного рядку для роботи з розподіленою файловою системою, основний сервер встановлюється на віддалений сервер і його розташування надається клієнту, сервери даних створюють кластер для зберігання реплік блоків.

4.1.2. Структура системи та виконувані функції

Зазвичай клієнтська частина використовується користувачем у вигляді сервісу для розробки програм, яким необхідна можливість зберігати файли, проте також може бути використана напряму завдяки інтерфейсу командного рядка. Клієнтська програма складається з двох частин: файлу налаштувань та функцій роботи із системою.

Основною роллю даних функій є робота з простором імен розподіленої файлової системи та збереження і отримання файлу. Розглянемо деякі з них:

1. Функції встановлення зв’язку з основним сервером та серверами даних на основі їх ip адреси та порту.
2. Функція збереження та читання файлу налаштувань, такі як робоча директорія, місце збереження файлів, або адреса основного сервера.
3. Команда інтерфейсу командного рядка для отримання директорії відносно якої відбувається маніпулювання простором імен. Ця функція розроблена за особливістю команди pwd в Linux.

4. Команда інтерфейсу командного рядка для виводу вмісту директорії, її піддиректорій і файлів. Дана команда подібна до команди ls в Linux.

5. Команда інтерфейсу командного рядка для виводу дерева директорій і файлів відносно заданого простору імен.

6. Команда інтерфейсу командного рядка для створення нової директорії. При створенні уже існуючої директорії система видасть помилку користувача.

7. Команда інтерфейсу командного рядка для переходу між різними директоріями. Вона схожа на команду cd системи Linux.

8. Команда інтерфейсу командного рядка для видалення директорій. В своїй структурі система не видаляє директорії, а переносить їх в приховану директорію, що дає можливість відновити файли.

9. Команда інтерфейсу командного рядка для запису файлу в систему. Вона приймає шлях до файлу та маж режим перезапису файлу. Функція складається з двох етапів: реєстрації простору імен в основному сервері і отримання локації для блоків файлів та запису блоків файлів на сервери даних.

10. Команда інтерфейсу командного рядка для читання файлу із розподіленої файлової системи. Функція отримує місце знаходження блоків файлу від основного серверу та зчитує їх із серверів даних. Файл зберігається в директорії локальної системи визначеної в файлі налаштувань.

Основний сервер розподіленої файлової системи відповідальний за маніпулювання даними про простір імен та сервери даних. Система повинна мати лише один основний сервер для вирішення проблем синхронізації, проте допускається створення тіньових серверів копій для забезпечення працездатності системи в разі помилки.

Основний сервер містить ряд методів для комунікації із клієнтом та серверами даних:

1. Методи збереження і завантаження метаданих системи із тимчасових файлів. Це потрібно для збереження даних в разі перезапуску системи, або відмови основного серверу.

2. Метод для розподілу чанків системи для реплік блоків в залежності від їх кількості.

3. Метод запису файлу. В ньому відбувається зміна простору імен та розподілення серверів даних для чанків.

4. Метод читання файлу. Цей метод повертає список локацій серверів даних для читання блоків файлу.

5. Метод отримання локацій чанків базуючись на їх унікальному ідентифікаторі.

6. Методи маніпулювання простором імен з якими взаємодіє користувач: створення директорії, отримання простору імен, видалення директорії, перевірка існування шляху директорії.

7. Методи взаємодії із серверами даних: реєстрування чанк серверів, оновлення даних та отримання їх місце знаходжень.

Сервери даних виконують роль збереження реплік блоків файлів і формують кластери навколо основного серверу. Різні сервери зберігають одні й ті самі репліки для збереження надмірності. Дані сервери виконуються пасивну роль збереження і відповіді на запит користувача, або інструкції основного серверу.

Розглянемо методи серверу даних:

1. Методи для комунікації із користувачем: запис та читання блоку файлу за допомогою його ідентифікатора. Сервери даних зберігають блоки файлів, як звичайні файли Linux.

2. Методи сканування стану своєї системи і звітування основному серверу про них.

3. Метод підключення до кластеру. Він реєструється в основному сервері.

4.1.3. Необхідні технічні засоби

Всі технічні засоби, які вимагаються для встановлення і роботи даної системи є безплатними та легкими до встановлення. Наведемо їх список:

- Linux (ОС для роботи системи)

- Python 3.7 (Система може працювати і на інших версія, проте без гарантії)
- PIP (Для встановлення необхідних модулів Python)
- Git (Зручний інструмент для отримання даної системи)
- grpc (Модуль Pyhton)
- click (Модуль Pyhton)

4.1.4. Вхідні та вихідні дані

В залежності від операцій система може мати різні вхідні та вихідні дані, наведемо їх для різних процесів:

- Запис файлу:
  - Вхідні:
    - Назва файлу
    - Розмір файлу
    - Шлях до файлу
    - Файл
    - Директорія для збереження
  - Вихідні:
    - Локації серверів даних для збереження блоків файлу
- Читання файлу:
  - Вхідні:
    - Простір імен файлу
    - Локальна директорія для збереження файлу
  - Вихідні:
    - Локації серверів даних для читання блоків файлу
    - Файл
- Маніпулювання із простором імен:
  - Вхідні:
    - Команда

- Простір імен

■ Вихідні:

- Зміна в просторі імен

4.2. Інструкція користувача

4.2.1. Вступ

Мета системи зберігання файлів в розподіленій архітектурі забезпечуючи зручне читання та надмірність для збереження файлів в разі помилок. Система розрахована для використання при розробці додатків, яким необхідне інструмент для зберігання даних, проте також може бути використаний напряму за допомогою інтерфейсу командного рядка. Система знаходиться на початковому етапу і вимагає додаткового розвитку для комерційного використання.

4.2.2. Концепт роботи

Система працює на базі Linux і встановлюється за допомогою пакетів Python. Сама система поділена на три частини:

- Клієнт - інтерфейс командного рядка
- Основний сервер - збереження метаданих
- Сервери даних - збереження файлів

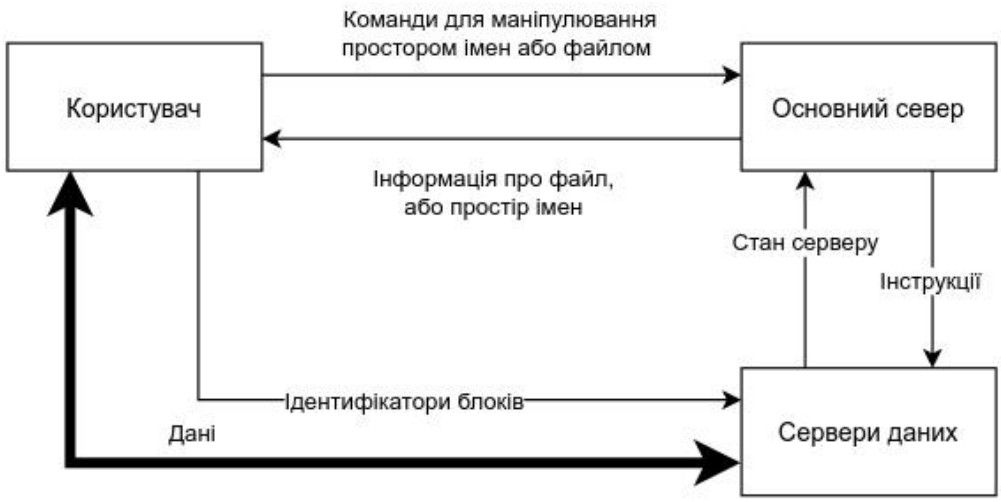

Рис.4.1 Концепт роботи розподіленої файлової системи

Концепт роботи зображений на Рис.4.1. Як показано на рисинку основне навантаження на мережу при передачі інформації припадає на передачу блоків файлів між клієнтом та сервером даних. Основний сервер не залучений в процес передачі великих об’ємів даних. Цей концепт дозволяє зробити основний сервер швидким і відповідати на багато паралельних, але легких запитів. В той час як сервери даних лише зберігають блоки файлу за командую користувача використовуючи унікальні ідентифікатори. Основний сервер також періодично комунікує із серверами даних для перевірки їх стану та надання інструкцій роботи в разі надзвичайних ситуацій.

4.2.3. Встановлення системи

Система розробляла на платформі Linux Debian Buster, проте може працювати на більшості популярних дистрибутивів. Інструкція встановлення системи:

1. Встановити Python 3.7 (Рис.4.2, Рис.4.3). Встановлення необхідну версію Python можна легко за допомогою команди apt-get командного рядка Linux. Є також альтернативний спосіб встановлення побудувавши виконуваний файл із вихідного коду [34].

```
king@debian:~$ sudo apt-get install python3.7
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Рис.4.2 Встановлення Python

```
king@debian:~$ python3 -V
Python 3.7.3
```

Рис.4.3 Перевірка правильності встановлення

2. Встановити pip (Рис.4.4, Рис.4.5). Інструмент використовується для встановлення необхідних модулів Python для роботи системи.

```
king@debian:~$ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Рис.4.4 Встановлення Pip

```
king@debian:~$ pip3 -V
pip 18.1 from /usr/lib/python3/dist-packages/pip (python 3.7)
```

Рис.4.5 Перевірка правильності встановлення

3. Встановити git (Рис.4.6, Рис.4.7). Даний інструмент необхідний для завантаження коду програми. Він також встановлюється за допомогою команди apt-get командного рядка Linux.

```
king@debian:~$ sudo apt-get install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Рис.4.6 Встановлення Git

```
king@debian:~$ git --version
git version 2.20.1
```

Рис.4.7 Перевірка правильності встановлення

4. Клонувати репозиторій (Рис.4.8).

```
king@debian:/tmp/diplom$ git clone https://github.com/ChaosGuru/DFSs.git
Cloning into 'DFSs'...
remote: Enumerating objects: 250, done.
remote: Counting objects: 100% (250/250), done.
remote: Compressing objects: 100% (160/160), done.
remote: Total 250 (delta 102), reused 217 (delta 69), pack-reused 0
Receiving objects: 100% (250/250), 7.74 MiB | 8.44 MiB/s, done.
Resolving deltas: 100% (102/102), done.
```

Рис.4.8 Клонування коду програми

5. Встановити необхідні пакети (Рис.4.9). В залежності від бажання користувача він може створити віртуальне середовище для пакетів Python.

```
(.venv) king@debian:/tmp/diplom/DFSs$ pip3 install rpyc==5.0.1 click==7.1.2
Collecting rpyc==5.0.1
  Using cached https://files.pythonhosted.org/packages/e8/4c/6d456ae4319190d17e0
d4cd8c1ee6b6ba9125f0bde18ef37afdb50867a39/rpyc-5.0.1-py3-none-any.whl
Collecting click==7.1.2
  Using cached https://files.pythonhosted.org/packages/d2/3d/fa76db83bf75c4f8d33
8c2fd15c8d33fdd7ad23a9b5e57eb6c5de26b430e/click-7.1.2-py2.py3-none-any.whl
Collecting plumbum (from rpyc==5.0.1)
  Using cached https://files.pythonhosted.org/packages/6c/fc/6cdaf59a001c7073338
69b054daf1e0df02978d261f20f8b082afcf189c3/plumbum-1.7.0-py2.py3-none-any.whl
Installing collected packages: plumbum, rpyc, click
Successfully installed click-7.1.2 plumbum-1.7.0 rpyc-5.0.1
```

Рис.4.9 Встановлення пакетів

Дана система може працювати і на інших версіях інструментів, проте коректність роботи системи не гарантується.

4.2.4. Інтерфейс командного рядка

Для використання клієнтського інтерфейсу командного рядка необхідно мати встановлений python. Для його використання достатньо запустити модуль client_kun.py в директорії client пакету програми (Рис.4.10) та задати відповідні параметри. Виклик модуля без параметрів покаже доступні команди.

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py
Usage: client_kun.py [OPTIONS] COMMAND [ARGS]...

  CLI for client DFS

Options:
  --help  Show this message and exit.

Commands:
  cd     Changes working directory
  get    Get file from DFSs
  ls     Prints directories and files
  mkdir  Creates new directory
  put    Put file to DFSs
  pwd    Prints current directory
  rm     Removes namespace
  tree   Print directory tree
```

Рис.4.10 Команди інтерфейсу командного рядка

Розглянемо команди інтерфейсу командного рядка:

1. pwd - виводить в консоль шлях директорії відносно якої відбуваються команди. Не потребує параметрів.

Лістинг.4.11 Команда pwd

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py pwd --help
Usage: client_kun.py pwd [OPTIONS]

  Prints current directory
```

2. cd - змінює директорію відносно якої виконуються маніпуляції над простором імен. Як параметр приймає шлях до директорії. Команда працює подібно до команди Linux, тому можливі параметри: ‘..’, ‘/’.

Лістинг.4.12 Команда cd

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py cd --help
Usage: client_kun.py cd [OPTIONS] PATH

  Changes working directory
```

3. ls - показує список директорій і файлів відносно робочої директорії, або може взяти за параметр іншу директорію і відобразити її вміст.

Лістинг.4.13 Команда ls

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py ls --help
Usage: client_kun.py ls [OPTIONS] [PATH]

  Prints directories and files
```

4. tree - відображає ієрархічну структуру файлової системи, її директорії і файли в робочий директорії. Може взяти за параметр шлях директорії і відобразити вміст в ній.

Лістинг.4.14 Команда tree

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py tree --help
Usage: client_kun.py tree [OPTIONS] [PATH]

  Print directory tree
```

5. mkdir - створює нову директорію в робочій директорії. Параметром виступає назва директорії, яка створюється.

Лістинг.4.15 Команда mkdir

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py mkdir --help
Usage: client_kun.py mkdir [OPTIONS] NAME

  Creates new directory
```

6. rm - видаляє певну директорію і його вміст. Параметром виступає простір імен для видалення.

Лістинг.4.16 Команда rm

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py rm --help
Usage: client_kun.py rm [OPTIONS] PATH

  Removes namespace
```

7. put - завантажує файл на розподілену файлову систему. Параметром є шлях до завантажуваного файлу.

Лістинг.4.17 Команда put

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py put --help
Usage: client_kun.py put [OPTIONS] FILENAME

  Put file to DFSs

Options:
  --force  Forces file writing.
```

8. get - отримати файл із розподіленої файлової системи. Параметром є назва файлу в просторі імен. Проте ідентифікатор файлу виступає його простір імен, тому файл можна отримати лише з директорії в якій він збережений.

Лістинг.4.18 Команда get

```
(.venv) king@debian:~/Coding/DFSs/client$ python3 client_kun.py get --help
Usage: client_kun.py get [OPTIONS] FILENAME

  Get file from DFSs
```

Команди інтерфейсу командного рядка були створені за схожими командами терміналу Linux.

4.3. Аналіз контрольного прикладу

Проведемо розгортання розподіленої файлової системи і продемонструємо її роботу. Для цього спочатку необхідно запустити головний сервер та кластер серверів даних (Рис.4.19).

```
(.venv) king@debian:~/Coding/DFSs$ python server/sensei_dono.py
DEBUG:sensei-dono:Creating sensei object
INFO:sensei-dono:Loading snapshot...
INFO:SENSEI/33333:server started on [0.0.0.0]:33333
```

Рис.4.19 Запуск основного серверу

Для розгортання кластеру серверів даних потрібно запустити файл start_cluster.sh в директорії data. Виконуваний файл по замовчуванню запускає 10 паралельних серверів даних на портах в діапазоні 30000-30010 (Рис.4.20).

```
(.venv) king@debian:~/Coding/DFSs/data$ ./start_cluster.sh
DEBUG:chunk-kun:Chunk server 30000 file system scan
DEBUG:chunk-kun:Chunk server 30000 greets sensei
DEBUG:chunk-kun:Chunk server 30002 file system scan
```

Рис.4.20 Запуск кластеру серверів даних

Дана система має можливість роботи на одній платформі. В даному випадку кластер серверів даних буде створений у вигляді директорій (Рис.4.21).

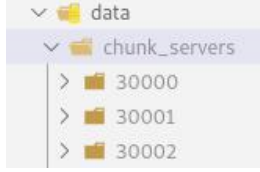

Рис.4.21 Кластер серверів даних

Продемонструємо режим роботи інтерфейсу командного рядка розподіленої файлової системи.

Команда pwd виводить робочу директорію, по замовчуванню це коренева директорія ‘/’ (Рис.4.22).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py pwd
/
```

Рис.4.22 Демонстрація роботи pwd

Для створення директорії використовується команда mkdir. Її створення відбувається в робочій директорії. Команда повертає новостворений простір імен (Рис.4.23).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py mkdir example
/example/
```

Рис.4.23 Демонстрація роботи

Вивід вмісту директорії відбувається за допомогою команди ls. По замовчуванню вона виводить вміст робочої директорії (Рис.4.24).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py ls
example
```

Рис.4.24 Демонстрація роботи ls

Щоб вивести весь простір імен використовується команда tree (Рис.4.25).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py tree
/example/
```

Рис.4.25 Демонстрація роботи tree

Зміна робочої директорії відбувається за допомогою команди cd. В разі не існування директорії користувача отримає відповідне повідомлення. Зміну робочої директорії можна перевірити за допомогою команди pwd (Рис.4.26).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py cd example2
Error! Path do not exists!
/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py cd example
/example/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py pwd
/example/
```

Рис.4.26 Демонстрація роботи cd

Перейшовши в іншу робочу директорію можна отримати список її піддиректорій та файлів, а також її весь простір імен (Рис.4.27).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py ls

(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py tree
/example/
```

Рис.4.27 Вміст робочої директорії

Команда cd дозволяє легко повернутися в кореневу директорію використавши параметр ‘/’ (Рис.4.28).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py cd /
/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py pwd
/
```

Рис.4.28 Демонстрація роботи cd

Розподілена файлова система надає можливість створення директорій всередині інших директорій. Для цього необхідно змінити робочу директорію і використати команду створення mkdir. За допомогою команд ls та tree можна перевірити вміст директорії (Рис.4.29).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py mkdir example2
/example2/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py tree
/example/
/example2/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py cd example2
/example2/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py mkdir new
/example2/new/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py mkdir new2
/example2/new2/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py ls
 new new2
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py tree
/example2/
/example2/new/
/example2/new2/
```

Рис.4.29 Створення директорій

Команда cd також надає зручне переміщення до батьківської директорії за допомогою параметра ‘..’ (Рис.4.30).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py pwd
/example2/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py cd ..
/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py pwd
/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py ls
example2 example
```

Рис.4.30 Демонстрація роботи cd

Команда ls також надає можливість перегляду вмісту інших директорій вказавши, як параметр відповідну назву директорії (Рис.4.31).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py ls example2
 new new2
```

Рис.4.31 Демонстрація роботи ls

Для видалення певного простору імен використовується команда rm. Їй потрібно вказати як параметр назву директорії для видалення. Використавши команду tree можна побачити, що видалення відбувається всього простору імен (Рис.4.32).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py tree
/example/
/example2/
/example2/new/
/example2/new2/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py rm example2
Removed 3 elements in namespace /example2/
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py tree
/example/
```

Рис.4.32 Демонстрація роботи

Для завантаження файлу в розподілену файлову систему використовується команда put. Для її завантаження необхідно вказати шлях до неї. Для тестування створимо невеликий файл (Рис.4.33).

```
(.venv) king@debian:/tmp$ cat test_dfss.txt
Text for saving in distributed file system.
```

Рис.4.33 Вміст тестового файлу

Збережемо файл в розподілену файлову систему вказавши шлях до нього. При зберіганні файлу він також записується в простір імен. Це можна перевірити за допомогою команд ls, або tree (Рис.4.34).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py put /tmp/test_dfs
s.txt
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py tree
/example/
/test_dfss.txt/
```

Рис.4.34 Збереження файлу

При запису файлу система отримує його розмір і розподіляє його на кількість блоків в залежності від заданого в системі максимального розміру блоків. Кожен

блок отримує репліку, яка буде записана на різних серверах даних. При запуску системи на одній платформі в відповідних директоріях будуть створені файли з даними блоку із розширенням .gfss (Рис.4.35).

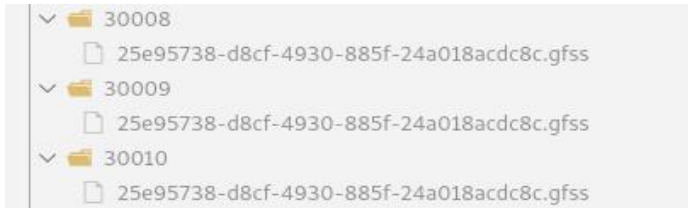

Рис.4.35 Репліки блоків на серверах даних

Для отримання файлу необхідно використати команду get і вказати назву файлу в робочій директорії (Рис.4.36).

```
(.venv) king@debian:~/Coding/DFSs/client$ python client_kun.py get test_dfss.txt
(.venv) king@debian:~/Coding/DFSs/client$
```

Рис.4.36 Отримання файлу

По замовчуванню отримуваний файл буде збережений в директорії saved_files (Рис.4.37).

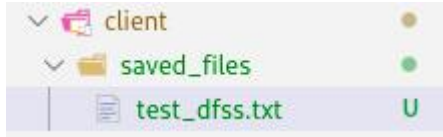

Рис.4.37 Отриманий файл

Перевіримо правильність читання файлу (Рис.4.38).

```
(.venv) king@debian:~/Coding/DFSs/client$ cat saved_files/test_dfss.txt
Text for saving in distributed file system.
```

Рис.4.38 Вміст файлу

Під час аналізу виконання контрольного прикладу помітно, що клієнт має можливість успішно маніпулювати простором даних, а саме створювати та видаляти директорії, виводити їхній вміст, а також зберігати файли і читати їх використовуючи лише інтерфейс командного рядку. Під час збереження файлу було створено його репліки і збережено на різних серверах даних.

Висновок до четвертого розділу

В даному розділі було описано програму, її структура; функції, які вона виконує на стороні клієнта, основного серверу, серверів даних; необхідні системні вимоги для функціонування системи; вхідні та вихідні дані в системі. Також було створено інструкцію користувача в якій описано концепт роботи системи, спосіб її встановлення та надано список команд інтерфейсу командного рядка для клієнта. Вкінці було розглянуто контрольний приклад розподіленої файлової системи: виконання команд маніпулювання із простором імен і збереження та читання файлу із системи використовуючи інтерфейс командного рядка клієнта.

РОЗДІЛ 5

Економічне обґрунтування доцільності роботи

5.1. Економічна характеристика програмного продукту

Метою розроблення бакалаврської кваліфікаційної роботи є розробка системи розподіленого зберігання файлів. Вона дозволить користувачам з доступу до інтернету надійно зберігати свої файли. Дана система є сервісом розробленим для використання іншими програмними продуктами, а не кінцевими користувачами.

Основними витратами даної системи є серверні ресурси. Найбільшою з них є покупка і заміна жорстких дисків. Проте архітектура даної системи дозволяє використання дешевих компонентів, що значно знизить витрати на обладнання. Також використання лише одного центрального серверу і багатьох серверів даних з репліками блоків файлів дозволить легко додавати, або віднімати сервери від системи. Легкість масштабованості системи значно зменшить майбутні витрати порівняно з іншими системами.

На сьогодні існує багато аналогів подібних систем таких як Google File System, Hadoop Distributed File System і інші. Вони займають значну частину світового ринку і стоять майже на недосяжному рівні для молодих компаній. Проте поширеність компаній на світовому ринку в деяких випадках може бути і мінусом. Адже їхня політика зосереджена на глобальних питаннях, в той час, як питання локального ринку можуть бути проігноровані. В цьому випадку розроблення даної системи спрямованої для українського ринку може бути успішним.

Результат аналізу показав, що розробка даної системи є доцільною, проте містить значні ризики та конкуренцію.

5.2. Інформаційне забезпечення та формування гіпотези щодо потреби розроблення програмного продукту

На сьогоднішній день близько 60% всього населення планети активно користується інтернетом, це близько 4,6 мільярди людей, і з часом це число буде тільки зростати. Вони використовують різноманітні додатки та сервіси, які

зберігають дані на віддалених серверах. Прикладами є близько пів мільйона вебсайтів, що створюються кожен день, 95 мільйонів фото, які завантажують в Instagram за 24 години, гігабайти даних для навчань нейронних мереж, 500 годин відео в YouTube щохвилини. І тенденції створення і зберігання даних тільки зростають.

Для надійного зберігання даних використовуються розподілені файлові системи. Найпопулярнішими з яких є Google File System (GFS). Вона переважно використовується в продуктах Google, таких як Google Drive для зберігання документів і інших файлів, YouTube для публікації відео та інших. В зв'язку з популярністю даних сервісів, їхня система повинна справлятися з великими обсягами даних, та великим постійним навантаженням, проте що більш головніше бути надійною, не допускати втрати даних та стабільною. Також існує аналог GFS - Hadoop Distributed File System (HDFS). Це система з відкритим кодом, і з подібною архітектурою до GFS, але з дещо іншим функціоналом. Головною відмінністю є те, що будь-хто може купити доступ до їхнього API і користуватися нею. Окрім наведених вище систем, існує також багато інших, переважно створених для певних випадків використання.

Основними споживачами розподілених файлових систем є розробники хмарних сервісів, вебсайтів, мобільних додатків і інші, яким потрібно зберігати дані. Їхніми вимогами є надійність даних, тобто система не має допускати втрати даних та працювати без відмов, доступ через інтернет та прозорість використання. Також є особливі випадки, коли замовнику потрібна розподілена система, яка підходить для їхніх вимог, наприклад мати швидке читання файлів, або швидкість читання неважлива, але повинна бути можливість зберігати надзвичайно великі файли.

Попри існування великої кількості аналогів, в тому числі компанії світового масштабу, розробка системи може бути доцільною для локального ринку.

5.3. Оцінювання та аналізування факторів зовнішнього та внутрішнього середовищ

Проведемо оцінювання та аналіз факторів зовнішнього та внутрішнього середовищ груп експертів.

Фактори зовнішні оцінюються за шкалою [-5;5], при цьому межі шкали відображають максимальний негативний та позитивний вплив факторів на організацію, 0 демонструє, що фактор впливає на організацію нейтрально.

Фактори внутрішні оцінюються за шкалою [0;5], при цьому 0 демонструє нерозвинутість, відсутність чи катастрофічний стан фактора внутрішнього середовища, оцінка 5 демонструє високий рівень розвитку даного фактора.

Сума вагомостей усіх факторів становить одиницю, тобто рівень вагомості для кожного фактора визначається за допомогою коефіцієнтів. Зважений рівень впливу факторів розраховується як добуток впливу фактора у балах та рівня вагомості.

Таблиця 5.1

Результати експертного оцінювання впливу факторів зовнішнього та внутрішнього середовищ

Фактори	**Середня експертна оцінка, бали**	**Середня вагомість факторів**	**Зважений рівень впливу, бали**
1	2	3	4
Фактори зовнішнього середовища			
Споживачі	4	0,11	0,44
Постачальники	0	0	0
Конкуренти	-5	0,1	-0,5
Державні органи влади	-2	0,05	-0,1
Інфраструктура	4	0,04	0,16
Законодавчі акти	-4	0,1	-0,4
Профспілки, партії та інші громадські організації	0	0	0
Система економічних відносин в державі	-2	0,11	-0,22

Продовження таблиці 5.1

1	2	3	4
Організації-сусіди	-2	0,01	-0,02
Міжнародні події	4	0,01	0,04
Міжнародне оточення	5	0,03	0,15
Науково-технічний прогрес	5	0,12	0,6
Політичні обставини	-2	0,06	-0,12
Соціально-культурні обставини	0	0	0
Рівень техніки та технологій	5	0,16	0,8
Особливості міжнародних економічних відносин	3	0,02	0,06
Стан економіки	3	0,08	0,24
Загальна сума		1	1,13
Фактори внутрішнього середовища			
Цілі	4	0,11	0,44
Структура	5	0,16	0,8
Завдання	3	0,07	0,21
Технологія	5	0,2	1
Працівники	3	0,21	0,63
Ресурси	2	0,25	0,5
Загальна сума		1	3,58

Зважений фактор зовнішнього середовища становить 1,13. Позитивними факторами є споживачі, інфраструктура, міжнародні події, міжнародне оточення, науково-технічний прогрес, рівень техніки та технологій, особливості міжнародно технічних відносин та стан економіки. Негативними є конкуренти, державні органи влади, законодавчі акти, система економічних відносин в державі, організації-сусіди, політичні обставини. При створенні проекту конкуренти та державна влада можуть стати перешкодами при створенні проекту.

Зважений рівень внутрішнього середовища становить 3,58. Найважливішим фактором є технології.

Результатом експертного оцінювання впливу факторів зовнішнього та внутрішнього середовищ стала потреба у продукті.

5.4. Формування стратегічних альтернатив

Зробимо вибір за першою (Рис.5.1) та другою (Рис.5.2) групою альтернативних стратегій розвитку.

Для першої групи стратегічних альтернатив критеріями поділу альтернативних стратегій розвитку є існуючий продукт та новий, а також супутні послуги.

Рис.5.1 Стратегічні альтернативи першої групи

Стратегія розроблення нового продукту (проектного рішення) характеризується створенням абсолютно нового програмного забезпечення, яке дає змогу вирішити новоутворені потреби людини, суспільства, економіки тощо.

Стратегія розвитку існуючого продукту (проектного рішення) означає модифікацію програмного забезпечення, його якісних характеристик.

Стратегія розвитку існуючого продукту (проектного рішення) з супутніми послугами означає пропонування на ринку модифікованого програмного забезпечення із додатковими послугами (встановлення, супроводження, коригування, адаптування до специфіки конкретного підприємства тощо).

Стратегія нового продукту (проектного рішення) з супутніми послугами означає розроблення нового програмного забезпечення та пропонування при його експлуатації додаткових послуг.

Серед наведених стратегії обрано стратегію *розвитку існуючого продукту з супутніми послугами*.

Для другої групи стратегічних альтернатив критеріями поділу альтернативних стратегій розвитку є існуючий ринок та продукт, новий ринок та продукт.

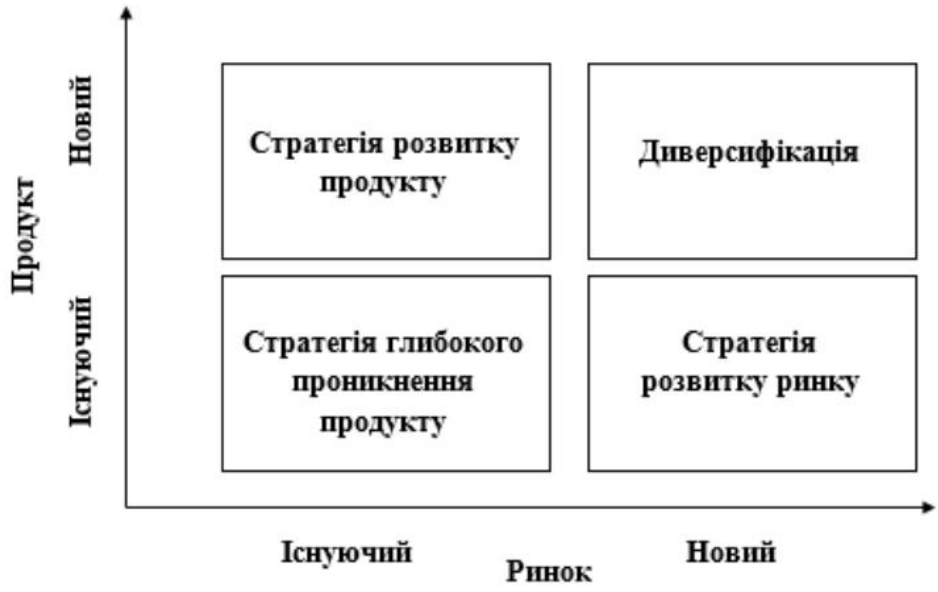

Рис.5.2 Стратегічні альтернативи другої групи

Глибше проникнення на ринок передбачає використання існуючого продукту для збільшення частки на існуючому ринку.

Стратегія розвитку ринку полягає в використанні існуючого продукту або незначній його модифікації для виходу на новий сегмент ринку, весь ринок або іноземний ринок.

Стратегія розвитку продукту полягає у створенні нового продукту для існуючого сегменту ринку.

Стратегія диверсифікації реалізується шляхом виходу на нові сфери бізнесу. Тобто розширення номенклатури товарів, послуг тощо.

Серед наведених стратегії обрано стратегію *диверсифікації*, в зв'язку з наданням нових послуг локальному ринку.

5.5. Бюджетування

Проведемо розрахунок витрат, пов'язаних з виготовленням та реалізацією продукту, яка дає можливість здійснити аналіз витрат. Визначимо собівартість

продукту та економічно обґрунтуємо доцільність вибору однієї з стратегій. Результати розрахунків наведемо в таблицях (Таблиці 5.2-5.8)

Таблиця 5.2

Бюджет витрат матеріалів та комплектуючих виробів

Назва матеріалів та комплектуючих	**Марка, тип, модель**	**Фактична кількість, шт.**	**Ціна за одиницю, грн.**	**Разом, грн.**
Сервер	Сервер SuperMicro SC-848A (X9QRi-F+)	1	115 868	115 868
Жорсткий диск	DELL 3.5" NLSAS 4TB 12Gbps	24	7 435	178 440
Маршрутизатор	Маршрутизатор Cisco RV340-K9-G5 RV340 Dual WAN Gigabit VPN Router	2	15 609	31 218
Разом:				**325 526**

Таблиця 5.3

Бюджет витрат на оплату праці

Посада, спеціальність	**Кількість працівників, осіб**	**Час роботи, дні**	**Денна заробітна плата працівників, грн.**	**Сума витрат на оплату праці, грн.**
Основна заробітна плата				
Архітектор	1	300	3 000	900 000
Програміст	2	300	1 500	450 000
Маркетолог	1	200	600	120 000
Разом:				**1 470 000**

Розрахуємо суму єдиного соціального страхування по ставці 22%, суму податку з доходів по ставці 18% та суму військового збору із зарплати по ставці 1,5%.

Таблиця 5.4

Бюджет обов'язкових відрахувань та податків

Посада, спеціальність	**Сума основної заробітної плати**	**Сума додаткової заробітної плати**	**Разом витрат на оплату праці**	**Сума єдиного внеску на соціальне страхування*, грн.**	**Сума податку з доходів фізичних осіб**, грн.**	**Сума податку військового збору із зарплати, грн**
Архітектор	900 000	0	900 000	198 000	162 000	13 500
Програміст	450 000	0	450 000	99 000	81 000	6 750
Маркетолог	120 000	0	120 000	26 400	21 600	1 800
Разом:	**1 470 000**	**0**	**1 470 000**	**323 400**	**264 600**	**22 050**

*нарахування єдиного внеску на соціальне страхування відповідає діючій ставці на момент виконання випускної роботи (22%);

**ставка податку з доходів фізичних осіб відповідає діючій на момент виконання випускної роботи (18%);

Розрахуємо бюджет загальновиробничих витрат в *Таблиці 5.5*.

Таблиця 5.5

Бюджет загальновиробничих витрат

Статті витрат	**Сума, грн.**
Змінні загальновиробничі витрати, у т.ч.:	
- заробітна плата допоміжного персоналу;	10 000
- витрати на МШП;	10 000
- витрати на електроенергію;	50 000
- витрати на ремонт;	20 000
- інші змінні витрати;	50 000
Разом змінних витрат:	140 000
Постійні загальновиробничі витрати, у т.ч.:	
- заробітна плата допоміжного персоналу;	10 000
- комунальні послуги;	10 000
- витрати на оренду;	15 000
- витрати на ремонт;	15 000
- інші постійні витрати;	20 000
Разом постійних витрат:	70 000
Разом загальновиробничих витрат:	**210 000**

Розрахуємо бюджет адміністративних витрат та витрат на збут в *Таблиці 5.6*.

Таблиця 5.6

Бюджет адміністративних витрат та витрат на збут

Статті витрат	**Сума, грн.**
Адміністративні витрати, у т.ч.:	
- заробітна плата адміністративного персоналу;	15 000
- витрати на МШП;	10 000
- витрати на сплату податків і зборів;	50 000
- знос адміністративного обладнання;	50 000
- інші адміністративні витрати;	15 000
Разом адміністративних витрат:	**140 000**
Витрати на збут, у т.ч.:	
- витрати на рекламу;	50 000
- інші витрати на збут;	20 000
Разом витрат на збут:	**70 000**

Запишемо зведений кошторис витрат на розробку проектного рішення в *Таблицю 5.7*.

Таблиця 5.7

Зведений кошторис витрат на розробку проектного рішення (продукту)

Статті витрат	**Одиниці виміру**	**Фактична кількість, шт.**	**Ціна одиниці, грн.**	**Разом, грн.**
Купівельні напівфабрикати та комплектуючі вироби	грн	-	-	325 526
Основна заробітна плата	грн	-	-	1 470 000
Додаткова заробітна плата	грн	-	-	-
Відрахування на соціальне страхування	грн	-	-	323 400
Витрати на утримання й експлуатацію устаткування	грн	-	-	50 000
Загальновиробничі витрати, у т.ч.:	грн	-	-	
- змінні;	грн	-	-	140 000
- постійні;	грн	-	-	70 000
Разом виробничих витрат:	грн	-	-	2 378 926
Адміністративні витрати	грн	-	-	140 000
Витрати на збут	грн	-	-	70 000
Інші операційні витрати	грн	-	-	-
Разом виробничих і операційних витрат:	**грн**	**-**	**-**	**2 588 926**

Розрахуємо фінансові результати розрахувавши вартість продукту, який розробляється.

Ціна проектного рішення - це сума собівартості до добутку собівартості на рентабельність:

2 378 926 + 2 378 926 * 0,49 = 3 544 599.74 (грн).

Податок на додану вартість:

3 544 599.74 * 0,2 = 708 919.95 (грн).

Чистий дохід від реалізації продукту:

3 544 599.74 - 708 919.95 = 2 835 679.8 (грн).

Валовий прибуток:

2 835 679.8 - 2 378 926 = 456 753.8 (грн).

Фінансовий результат від операційної діяльності:

456 753.8 - 210 000 = 246 753.8 (грн).

Податок на прибуток:

246 753.8 * 0,18 = 44 415.68 (грн).

Чистий прибуток (збиток):

246 753.8 - 44 415.68 = 202 338.12 (грн).

Таблиця 5.8

Бюджет фінансових результатів

Показники	**Сума, грн.**
Дохід від реалізації продукції	3 544 599.74
Податок на додану вартість***	708 919.95
Чистий дохід від реалізації продукції	2 835 679.8
Собівартість реалізованої продукції	2 378 926
Валовий прибуток	456 753.8
Операційні витрати:	
- адміністративні витрати:	140 000
- витрати на збут;	70 000
- інші операційні витрати;	-
Фінансовий результат від операційної діяльності	246 753.8
Податок на прибуток ****	44 415.68
Чистий прибуток (збиток)	**202 338.12**

***ставка податку на додану вартість відповідає діючій на момент виконання випускної роботи (20%);

****ставка податку на прибуток відповідає діючій на момент виконання випускної роботи (18%);

Висновок до п’ятого розділу

На основі попередніх розділів було вибрано стратегію розвитку існуючого продукту з супутніми послугами, оскільки на ринку знаходиться сильні конкуренти проте є можливість розвитку продукту на локальному ринку. Також потреба у подібних системах щороку зростає разом із збільшенням використання хмарних послуг, що дасть нові можливості для нових компаній.

Головною перевагою розроблення даного продукту стане надання переваги сервісам для українського ринку. Попри значну конкуренцію великих компаній, їхні політики не завжди діють найкращим чином для локального ринку, тому сервіси нашої системи будуть розроблені з увагою на потреби наших користувачів. Також розміщення серверів в Україні значно зменшить вартість надання послуг, а також збільшить швидкість обміну даними, в зв’язку з своєю близькістю до користувачів.

ВИСНОВКИ

В результаті виконання бакалаврської кваліфікаційної роботи було розроблено розподілену файлову систему зберігання файлів, яка дозволяє безпечно зберегти дані на віддаленому пристрої. Проаналізовано відомі підходи до розробки архітектури подібних систем, а також можливі їх покращення. Проведено системний аналіз, розроблено дерево цілей та вибрано найкращу альтернати системи. Проведено аналіз вибраних програмних засобі та наведено їх технічні характеристики. Створено систему та показано її роботу. Провели економічні підрахунки для підтвердження доцільності створення даного продукту.

В першому розділи роботи було проаналізовано літературні та електронні джерела архітектур, розвитку та недоліків розподілених файлових систем. Ринок розподілених систем розвивався уже кілька десятків років і володіє різноманітними архітектурами і реалізаціями. Більшість з них володіють такими спільними характеристиками: прозорості - дозволяє проводити операції із файлами наче вони на локальному комп’ютері та надмірності - забезпечує цілісність файлів та їх відновлення в разі помилки системи.

В другому розділі побудована архітектуру інформаційної системи розподіленого зберігання файлів. Проведений системний аналіз, побудована дерево цілей та за допомогою аналізу ієрархій вибрано краще альтернативу системи. За допомогою програми AllFusion Process Modeler було побудовано контекстну діаграму та її декомпозицію. Також, за допомогою цієї ж програми побудовано ієрархію задач.

В третьому розділі роботи було вибрано програмні засоби та наведено їх технічні альтернативи. Для побудови системи було вибрано наступні програмні засоби: мову програмування загального призначення Python та платформу для роботи системи Linux. Комунікація в системі здійснюється за допомогою технології RPC. Середовище розробки VSCode.

В четвертому розділі було описано програму та її структуру. Проаналізовано функції, які виконуються на стороні клієнта, основного сервера та серверів даних.

Наведено необхідні системні вимоги для роботи та подано вхідні і вихідні дані в системі. Створено інструкцію користувача в якій описано концепт роботи системи, спосіб її встановлення та надано список команд інтерфейсу командного рядка для клієнта. На контрольному прикладі продемонстровано роботу системи: маніпулювання із простором імен і збереження та читання файлу із системи використовуючи інтерфейс командного рядка клієнта.

В п’ятому розділі економічно обґрунтовано доцільність виконання проекту та проаналізовано економічний вплив на розвиток продукту. В результаті було вибрано стратегію розвитку існуючого продукту з супутніми послугами, складено кошторис, оцінено вплив внутрішніх та зовнішніх факторів та проаналізовано аналогічні продукти. Також побудовано бюджет фінансових результатів і знайдено чистий прибуток.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Google File System [Електронний ресурс] / Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung Google – Режим доступу: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/035fc972c796d33122033a0614bc94cff1527999.pdf (дата звернення 05.03.2021).

2. What is DFS(Distributed File System)? [Електронний ресурс] / Snigdha Yambadwar - Режим доступу: https://www.geeksforgeeks.org/what-is-dfsdistributed-file-system/ (дата звернення 06.03.2021).

3. Distributed File System (DFS) [Електронний ресурс] / techopedia - Режим доступу: https://www.techopedia.com/definition/1825/distributed-file-system-dfs (дата звернення 07.03.2021).

4. Evolution and Analysis of Distributed File Systems in Cloud Storage: Analytical Survey [Електронний ресурс] / Dharavath Ramesh, Neeraj Patidar, Gaurav Kumar, Teja Vunnam — Режим доступу: https://www.researchgate.net/publication/312469756_Evolution_and_Analysis_of_Distributed_File_Systems_in_Cloud_Storage_Analytical_Survey (дата звернення 08.03.2021).

5. Network File System (NFS) [Електронний ресурс] / TechTarget — Режим доступу: https://searchenterprisedesktop.techtarget.com/definition/Network-File-System (дата звернення 10.03.2021).

6. Network File System (NFS) [Електронний ресурс] / GeekForGeeks — Режим доступу: https://www.geeksforgeeks.org/network-file-system-nfs/ (дата звернення 11.03.2021).

7. Mirjana Spasojevic. An empirical study of a wide-area distributed file system / Mirjana Spasojevic, Mahadev Satyanarayanan - ACM Transactions on Computer SystemsMay 1996 (дата звернення 12.03.2021).

8. You really should know what the Andrew File System is [Електронний ресурс] / Bob Brown, Network World - https://www.networkworld.com/article/3195838/you-really-should-know-what-the-andrew-file-system-is.html (дата звернення 13.03.2021).

9. Google File System (GFS) [Електронний ресурс] / techopedia - Режим доступу: https://www.techopedia.com/definition/26906/google-file-system-gfs (дата звернення 14.03.2021).

10. Hadoop: что, где и зачем [Електронний ресурс] / Хабр, @ffriend – Режим доступу: https://habr.com/ru/post/240405/ (дата звернення 15.03.2021).

11. 7 основных преимуществ и пара недостатков Apache HBase для Big Data систем [Електронний ресурс] / Анна Вичугова, Big Data School - Режим доступу: https://www.bigdataschool.ru/blog/hbase-use-cases.html (дата звернення 17.03.2021).

12. Centralized Cache Management in HDFS [Електронний ресурс] / The Apache Software Foundation — Режим доступу: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/CentralizedCacheManagement.html (дата звернення 20.03.2021).

13. Метод аналізу ієрархій [Електронний ресурс] / Режим доступу: https://dss.tg.ck.ua/ahp-help (дата звернення 21.03.2021).

14. J Blomer. A Survey on Distributed File System Technology / J Blomer — CERN 2015 (дата звернення 22.03.2021).

15. What is Python? Executive Summary [Електронний ресурс] / Режим доступу: https://www.python.org/doc/essays/blurb/ (дата звернення 25.03.2021).

16. Thain D. and Livny M. Scalable Computing: Practice and Experience, 2005 (дата звернення 26.03.2021).

17. A Full Introduction to DFS (Distributed File System) [Електронний ресурс] / MiniTool — Режим доступу: https://www.minitool.com/lib/distributed-file-system.html (дата звернення 28.03.2021).

18. WSEAS Transactions on Computers 4 / [Dorigo A, Elmer P, Furano F, Hanushevsky A], 2005. - С. 348–353 (дата звернення 30.03.2021).

19. Proc. of the 26th IEEE Sympoisum on Mass Storage and Technologies (MSST'10) / [Shvachko K, Kuang H, Radia S, Chansler R], 2010. - С. 1-10 (дата звернення 02.04.2021).

20. Journal of Physics: Conference Series / [Blomer J, Aguado-Sanchez C, Buncic P, Harutyunyan A], 2011 (дата звернення 03.04.2021).

21. Schwan P. Proc. of the 2003 Linux Symposium / 2003. - С. 380–386 (дата звернення 05.04.2021).

22. Filesystem in Userspace (FUSE) [Електронний ресурс] / Henk C., Szeredi M. — Режим доступу: http://fuse.sourceforge.net (дата звернення 06.04.2021).

23. Comparing Python to Other Languages [Електронний ресурс] / Режим доступу: https://www.python.org/doc/essays/comparisons/ (дата звернення 07.04.2021).

24. System Analysis and Design — Overview [Електронний ресурс] / Режим доступу: https://www.tutorialspoint.com/system_analysis_and_design/system_analysis_and_design_overview.htm (дата звернення 09.04.2021).

25. Catter McNamara. Field Guide to Consulting and Organizational Development / Catter McNamara // MBA — 140с (дата звернення 12.04.2021).

26. What is a Goal Tree? [Електронний ресурс] / Chris Hohmann — Режим доступу: https://hohmannchris.wordpress.com/2014/03/07/what-is-a-goal-tree/ (дата звернення 15.04.2021).

27. Remote Procedure Call (RPC) [Електронний ресурс] / Режим доступу: https://www.tutorialspoint.com/remote-procedure-call-rpc (дата звернення 16.04.2021).

28. RPyC - Transparent, Symmetric Distributed Computing [Електронний ресурс] / Режим доступу: https://rpyc.readthedocs.io/en/latest/ (дата звернення 18.04.2021).

29. Python Features [Електронний ресурс] / Режим доступу: https://www.geeksforgeeks.org/python-features/ (дата звернення 22.04.2021).

30. Some Limitations of Python [Електронний ресурс] / Режим доступу: https://medium.com/@mindfiresolutions.usa/some-limitations-of-python-6f5370fc215f (дата звернення 25.04.2021).

31. Theory of Operation (RPyC) [Електронний ресурс] / Режим доступу: https://rpyc.readthedocs.io/en/latest/docs/theory.html#theory (дата звернення 26.04.2021).

32. 111+ Linux Statistics and Facts – Linux Rocks! [Електронний ресурс] / Nick Galov - Режим доступу: https://hostingtribunal.com/blog/linux-statistics/#gref (дата звернення 27.04.2021).

33. Linux Server Architecture & its Features [Електронний ресурс] / Режим доступу: https://www.snwntechsolution.com/blog/linux-server-architecture-its-features/ (дата звернення 28.04.2021).

34. How to Install Python on Debian 10 [Електронний ресурс] / Karim Buzdar Режим доступу: https://linuxhint.com/install-python-debian-10/ (дата звернення 29.04.2021).

АНОТАЦІЯ

Нанівський О.І., Рішняк І.В. (керівник). Інформаційна система розподіленого зберігання файлів. Бакалаврська кваліфікаційна робота. - Національний університет «Львівська політехніка», Львів, 2021.

Сьогодні існує багато розподілених файлових систем з різними архітектурами для різних потреб з своїми перевагами та недоліками, оскільки вони можуть значно спростити збереження і обробку великих даних. Переважно це громіздкі системи, які забезпечують високу продуктивність. Проте вони мають складну архітектуру та забирають багато часу і ресурсів для імплементації.

В цій роботі представлена розподілена файлова система з простою архітектурою, яка забезпечує легке встановлення і налаштування. В ній реалізовано процеси зберігання файлів, їх читання та маніпуляція простором імен. Дана система забезпечує прозорість використання, коли користувач не знає про архітектуру системи та спосіб зберігання файлів, а також її надмірність, щоб забезпечує автоматичне відновлення роботи системи та втрачених файлів.

Дана система надихалася архітектурними особливостями DFS, а також її аналогом з відкритим кодом HDFS [1,2]. Дані системи мають просту та ефективну архітектуру побудовану використовуючи досвід зберігання даних в Google та інших великих компаніях.

Побудову архітектури розподіленої файлової системи було зроблено за допомогою діаграми потоків даних, а також деталізовано процес зберігання файлу. Архітектура системи містить три основні елементи: клієнта, основний сервер та сервери даних. Основне навантаження потоку даних відбувається між клієнтом та серверами даних, основний сервер не бере участі в безпосередньому збережені даних, проте він відповідає за швидке збереження і отримання метаданих про файли та його блоки.

Дана система розроблялася і призначена працювати на операційній системі Linux. Це дозволить встановити програму на більшості віддалених серверів. Код програми написаний за допомогою мови загального призначення Python, що надає

усі інструменти для маніпулювання із операційною системою, а також віддаленого комунікування між клієнтом та віддаленими серверами. На стороні клієнта розроблений інструмент комунікації із системою - інтерфейс командного рядку, команди якого мають назву і схожий функціонал, як команди Linux.

Розроблена система забезпечує надійне зберігання даних за допомогою надмірності - репліки блоків і зберігання їх на різних серверах та прозорості - клієнт не знає про деталі внутрішньої архітектури системи і використовує файли як на локальній файловій системі. Також проста архітектура дозволяє легко та швидко розгортати систему.

Ключові слова - розподілена файлова система, DFS, HDFS, прозорість, надмірність, Linux.

Перелік використаних літературних джерел.

1. The Google File System [Електронний ресурс] / Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung Google – Режим доступу: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/035fc972c796d33122033a0614bc94cff1527999.pdf .

2. Centralized Cache Management in HDFS [Електронний ресурс] / The Apache Software Foundation — Режим доступу: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/CentralizedCacheManagement.html .

ABSTRACT

Nanivsky O.I, Rishnyak I.V. Information system for distributed file storage. Bachelor's thesis. - Lviv Polytechnic National University, Lviv, 2021.

Today, there are many distributed file systems with different architectures for different needs with their advantages and disadvantages, as they can greatly simplify the storage and processing of large data. Mostly they are large systems that provide high performance. However, they have a complex architecture and take a lot of time and resources to implement.

This paper presents a distributed file system with a simple architecture that provides easy installation and configuration. It implements the processes of storing files, reading them and manipulating the namespaces. The system provides transparency of use, which means user is unaware of the system architecture and method of file storage, as well as its redundancy, to ensure automatic recovery of the system and lost files.

This system was inspired by the architectural features of DFS, as well as its open source analogue - HDFS [1,2]. These systems have a simple and efficient architecture which is built using the huge experience of Google and other large companies.

The architecture of the distributed file system was built using a data flow diagram, and the process of storing the file was detailized. The system architecture consist of three main elements: the client, the main server and the data servers. The main load of the data flow occurs between the client and the data servers, the main server is not involved in the data storing, but it is responsible for the rapid storage and retrieval of metadata about the files and its blocks.

This system was developed and designed to run on the Linux operating system. This will allow you to install the program on most remote servers. The program code is written in the general-purpose Python language, which provides all the tools for manipulating the operating system, as well as remote communication between the client and remote servers. On the client side, a tool for communication with the system has been developed - a command line interface, the commands of which have a name and similar functionality to Linux commands.

The developed system provides reliable storage of data by means of redundancy - replicas of blocks which are seved on different servers and transparency - the client does not know details about internal architecture of system and uses files as on local file system. Also, the simple architecture allows you to easily and quickly deploy the system.

Keywords: distributed file system, DFS, HDFS, transparency, redundancy, Linux.

References.

1. The Google File System [Електронний ресурс] / Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung Google – Режим доступу: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/035fc972c796d33122033a0614bc94cff1527999.pdf .

2. Centralized Cache Management in HDFS [Електронний ресурс] / The Apache Software Foundation — Режим доступу: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/CentralizedCacheManagement.html .