УДК 004.773:004.051(045)
Мельник В.М. к.ф-м.н. доц.
Луцький національний технічний університет

ВИКОРИСТАННЯ ЗВИЧАЙНИХ СОКЕТІВ API ДЛЯ ФАЙЛОВИХ СИСТЕМ ТИПУ HADOOP

Мельник В.М. Використання звичайних сокетів API для файлових систем типу Hadoop. Hadoop – це реалізація (каркас) для програмування розподілених обчислень з відкритим вихідним кодом яка базується на моделі MapReduce. Вона спочатку інтегрується з Hadoop Distributed File System (HDFS), тобто розподіленою файловою системою на рівні користувача. У статті описується оболонка-агностик API для файлової системи Hadoop (HFAA), щоб дати змогу їй інтегруватися з будь-якою розподіленою файловою системою через TCP-сокетну взаємодію. За допомогою такого API HDFS можна замінити будь-якою розподіленою файловою системою, такою як PVFS, Ceph, Lustre або іншою, дозволяючи прямим способом здійснювати співставлення з точки зору періодизації продуктивності і масштабованості. На відміну від попередніх спроб підвищення взаємодії системи Hadoop з новими файловими системами представлений сокет API усуває необхідність налаштування Hadoop-реалізації для Java і повертає реалізацію обов'язків її на саму файлову систему. Таким чином, розробники, бажаючі інтегрувати їх нову (іншу) файлову систему з Hadoop не несуть відповідальності за розуміння деталі внутрішньої роботи Hadoop в під'єднаній файловій системі.

В даному випадку початкова реалізація HFAA використовується для заміни HDFS файловою системою PVFS, популярною в високопродуктивних обчислювальних середовищах. Порівняно з альтернативним методом інтеграції з PVFS (інтерфейсу POSIX-ядра), HFAA збільшує пропускну здатність зпису і читання.

Ключові слова: Сокет API, Hadoop файлова система , розподілена обчислювальна реалізація Hadoop в Java

Мельник В.М. Использование обычных сокетов API для файловых систем типа Hadoop. Hadoop – это реализация (каркас) для программирования распределенных вычислений с открытым исходным кодом, основанная на модели MapReduce. Она сначала интегрируется с Hadoop Distributed File System (HDFS), т.е. распределенной файловой системой на уровне пользователя. В статье описывается оболочка-агностик API для файловой системы Hadoop (HFAA), чтобы дать возможность ей интегрироваться с любой распределенной файловой системой через TCP-сокетное взаимодействие. С помощью такого API HDFS можно заменить любой распределенной файловой системой, такой как PVFS, Ceph, Lustre или другой, позволяя прямым способом осуществлять сопоставление с точки зрения периодизации производительности и масштабируемости. В отличие от предыдущих попыток повышения взаимодействия системы Hadoop с новыми файловыми системами представлен API-сокет, устраняющий необходимость настройки Hadoop-реализации для Java и возвращающий реализацию ее обязанностей на саму файловую систему. Таким образом, разработчики, желающие интегрировать их новую (другую) файловую систему с Hadoop не несут ответственности за понимание детали внутренней работы Hadoop в подключенной файловой системе.

В данном случае начальная реализация HFAA используется для замены HDFS файловой системой PVFS, популярной в высокопроизводительных вычислительных средах. По сравнению с альтернативным методом интеграции с PVFS (интерфейса POSIX-ядра), HFAA увеличивает пропускную способность зписи и чтения.

Ключевые слова: Сокет API, Hadoop файловая система, распределенная вычислительная реализация Hadoop в Java

Melnyk V. Using a conventional socket API for hadoop file systems. Hadoop is an implementation (frame) for distributed computing programming with open source code, based on MapReduce model. It initially is integrated with the Hadoop Distributed File System (HDFS) at the user level. In this article the Hadoop Filesystem Agnostic API (HFAA)is introduced to allow Hadoop to integrate with any distributed file system over TCP sockets. With this API, HDFS can be replaced by distributed file systems such as PVFS, Ceph, Lustre, or others, thereby allowing direct comparisons in terms of performance and scalability. Unlike previous attempts at augmenting Hadoop with new file systems, the socket API presented here eliminates the need to customize Hadoop's Java implementation, and instead moves the implementation responsibilities to the file system itself. Thus, developers wishing to integrate their new file system with Hadoop are not responsible for understanding details of Hadoop's internal operation.

In this case, an initial implementation of HFAA is used to replace HDFS with PVFS, a file system popular in highperformance computing environments. Compared with an alternate method of integrating with PVFS (a POSIX kernel interface), HFAA increases write and read throughput.

Keywords: socket API, Hadoop file system, Distributed computing, Hadoop's Java implementation

Вступ

До складу ***Hadoop Common*** входять библиотеки управління файловими системами, що підтримуються Hadoop, сценарії створення необхідної інфраструктури та управління розподіленою обробкою даних, для вигідності виконання яких створений спеціалізований спрощений інтерпретатор командного рядка, що стартує із оболонки операційної системи через відповідну команду.

Hadoop MapReduce являється програмним каркасом для програмування розподілених обчислень в рамках парадигми MapReduce [1]. Розробнику додатка для Hadoop MapReduce

необхідно реалізувати базовий обробник, який на кожному вузлі обчислення кластера забезпечує перетворення вихідних пар «ключ-значення» в проміжний набір пар «ключ – значення» (клас, який реалізує інтерфейс Mapper, носить ім'я за функцією вищого порядку Map), і обробник, який зводить проміжний набір пар в результуючий, скорочений набір (клас, що реализує інтерфейс Reducer). Каркас передає на вхід такого скорочення відсортовані виведення з базових обробників. Зведення складається з трьох фаз: *shuffle* (*тасування*, виділення потрібної секції виводу), *sort* (*сортування*. Групування за ключами виведення від розподільників до сортування, яке необхідне в випадку, коли різні атомарні обробники повертають набір із однаковими ключами. При цьому, правила сортування на даній фазі можуть бути задані програмно і використовувати будь-які особливості внутрішньої структури ключів) і, власне, *reduce* (скоросення списку) – для отримання результуючого набору. Для деяких видів обробки стиснення не потрібне, і каркас повертає в цьому випадку набір відсортованих пар, отриманих базовими обробниками.

Hadoop MapReduce дозволяє створювати завдання як з базовими обробниками, так і зі стисненнями [1,2], написаними без використання Java: утиліти *Hadoop streaming* дозволяють використовувати в якості базових обробників і стиснень (зверток) будь-який файл виконання, який працює зі стандартним вводом-виводом операційної системи (наприклад, утиліти командної оболонки UNIX), є також SWING-сумістимий прикладний інтерфейс програмування *Hadoop pipes* на C++. Також, в склад дистрибутивів Hadoop входять реалізації різних конкретних базових обробників і зверток, що найбільш типічно використовуються в розподіленій обробці даних.

В перших версіях Hadoop MapReduce включав спеціальний засіб планування завдань (*JobTracker*) і вже з версії 2.0 ця функція перенесенв в YARN. З цієї версії модуль Hadoop MapReduce реалізований поверх YARN. Програмні інтерфейси в більшості збережені, однак повна зворотня сумісність не досягнута (тобто для запуску программ, написаних для попередніх версій API для роботи в YARN в загальному потрібна їх модифікація або рефракторинг, і тільки при певних обмеженнях можливі варіанти зворотньої двійкової сумісності.

HDFS (*Hadoop Distributed File System*) – це файлова система, призначена для збереження файлів великих розмірів, поблоково розподілених між вузлами кластера обчислення. Всі блоки в HDFS (крім останнього блоку файла) мають одинаковий розмір, і кожен блок може бути розміщений на декількох вузлах, розмір блоку і коефіцієнт реплікації (кількість вузлів, на яких повинен бути розміщений кожен блок) визначаються в налаштуваннях на рівні файла. Завдяки реплікації забезпечується стійкість розподіленої системи до збоїв окремих вузлів. Файли в HDFS можуть бути записані лише один раз (модифікація не підтримується), а запис в файл в одну мить часу може вести тільки один процес. Організація файлів в просторі імен – ієрархічна: наявний кореневий каталог, підтримується створення підкаталогів, в одному каталозі можуть бути розміщені файли і підкаталоги.Одже, Hadoop це базова основа з відкритим кодом, реалізуюча модель MapReduce для паралельного програмування [1,2]. Каркас Hadoop складається з двигуна MapReduce і файлової системи на рівні користувача. Для переносимості і простоти установки обидва компонента написані на Java і вимагають всього лиш стандартного обладнання. В останні роки, Hadoop став популярним в промисловості та академічних колах [3]. Наприклад, на кінець 2010 року, Yahoo було більш ніж 43 000 вузлів, що працюють на Hadoop як для застосування в області досліджень так і у виробництві [4].

HDFS управляє ресурсами зберігання в кластері Hadoop, забезпечуючи глобальний доступ до будь-якого файлу [5,6] і реалізується двома службами: один центральний вузол – NameNode і багатьох вузлів – DataNodes. NameNode несе відповідальність за підтримання дерева каталогів HDFS. Клієнти звертаються до NameNode для виконання спільних операцій файлової системи, таких як відкриття, закриття, перейменування і видалення. NameNode не зберігає власних даних HDFS, а скоріше всього підтримує процес відображення між іменем файлу HDFS і вузлом NameNode, список блоків у файлі, і вузлом/вузлами DataNode, на якому/яких ці блоки зберігаються.

Хоча HDFS зберігає файл даних в розподіленому режимі, файл метаданих зберігається за допомогою централізованої служби вузла NameNode. Як це властиво для дрібних кластерів, ця конструкція запобігає Hadoop від масштабування за рахунок ресурсів єдиного вузла NameNode. Попередній аналіз CPU-процесора і необхідної пам'яті для NameNode виявив, що ця служба має обмеження в пам'яті. Великий NameNode, наприклад, з 60 GB оперативної пам'яті може зберігати

не більше 100 мільйонів файлів в середньому розміром 128 MB (еквівалент двом HDFS блокам). Крім того, з 30% цільовим навантаженням на процесор CPU для служби малої затримки подібний NameNode може підтримати кластер з 100 000 користувачів для зчитування, проте тільки 10 тисяч користувачів з операцією запису [7].

За покращення масштабованості (цю ціль також розділяє HDFS "Федерація", наближення з використанням декількох незалежних просторів імен, що зберігаються на окремих NameNodes [8]), існують й інші мотивації заміни HDFS. Наприклад, альтернативні файлові системи дозволяють операції запису і перезапису будь де (підвищення гнучкості додатків) і підтримують віддалені пересилання для віддалених додатків через мережу як необмежену смугу (збільшення продуктивності передачі даних). Поряд з масштабованістю, продуктивністю і набором функцій підтримки, одним з ключових драйверів для використання альтернативних файлових систем є лише те, що ці файлові системи вже широко розгорнуті. У багатьох високопродуктивних обчислювальних середовищах Hadoop (тобто реалізація MapReduce) є найновішою реалізацією у найрізноманітніших підходах програмування додатків, які будуть підтримуватися з акцентом на компетентність існуючої інфраструктури.

Останні роботи описують досліджували заміни HDFS з іншими розподіленими файловими системами. На сьогодні Hadoop інтегрований з Amazon S3 [9], CloudStore [10], Ceph [11], GPFS [12], Lustre [13] і PVFS [14]. Кожна з цих реалізацій вимагає код спеціального призначення для інтеграції з Hadoop, і вимагає від розробника необхідності знання внутрішньої діяльності як Hadoop так і цільової файлової системи.

В даній статті дається оцінка трьом різним підходам заміни HDFS на інші розподілені файлові системи. В першому методі використовується драйвер POSIX для безпосереднього вмонтування розподіленої файлової системи на кожному вузлі кластера. Цей драйвер зазвичай надається з файловою системою і призначений для дозволу легкої інтеграції з будь-якою немодифікованою програмою або утилітою. Однак, при використанні Hadoop страждає продуктивність зберігання системи через обмеження інтерфейсу POSIX. Наприклад, Hadoop не в змозі запросити файлову систему для отримання інформації місця знаходження файлу, і, таким чином, не може спланувати розрахунки щоб мінімізувати передачу мережевих даних.

Другий підхід, код співвставки, що розширює реалізацію Hadoop Java (шляхом забезпечення модифікації абстрактного класу зручної файлової системи системою Hadoop) для прямого розташування іншої файлової системи в просторі користувача. Такий співвкладений код є специфікою конкретної файлової системи, тобто вставка для PVFS працює тільки з PVFS-системою.

Враховуючи вищеописаніі обмеження, третій спосіб включає розроблений агностик API файлової системи Hadoop (Hadoop Filesystem Agnostic API – ***HFAA***) для подолання масштабованих обмежень HDFS і надання можливості прямої інтеграції з альтернативними файловими системами. HFAA забезпечує універсальний, загальний інтерфейс, який дозволяє Hadoop співпрацювати з будь-якою файловою системою, підтримуючу мережеві сокети, а, особливо, файлові системи без єдиної точки відмови і більшою загальною цілеспрямованістю семантики. Його конструкція переміщає обов'язки інтеграції за межі Hadoop, і не вимагає від користувача або розробника базових знань framework для системи Hadoop MapReduce. По суті, інтеграція Hadoop зводиться до одного API.

Архітектура Hadoop

Framework для системи Hadoop реалізована у вигляді двох основних служб: двигуна Hadoop MapReduce і HDFS. Вони, як правило, використовується спільно, хоча при бажанні кожен з них може працювати незалежно. Наприклад, користувачі сервісу Amazon Elastic MapReduce можуть використовувати драйвер Hadoop MapReduce в поєднанні з власним сервісом збереження для Amazon (S3) [9].

В системі Hadoop, драйвер MapReduce реалізується двома програмними сервісами: робочий трекер (JobTracker) і трекер завдань (TaskTracker). Централізований сервіс JobTracker'а працює на виділеному вузлі кластера і несе відповідальність за розщеплення вхідних даних на частини для обробки незалежною картою і зменшенням задач (шляхом координації на рівні користувача з файловою системою), планування кожного завдання на вузлі кластера для його виконання, моніторингу процесу виконання, отримуючи сигнали підтвердження від вузлів кластера, і

відновлення після збоїв за допомогою повторного запуску завдань. На кожному вузлі кластера примірник служби TaskTracker'а отримує карту, знижує задачі з JobTracker'а, виконує завдання і повідомляє про цей стан назад у JobTracker.

HDFS забезпечує глобальний доступ до будь-якого файлу в розділювальному просторі імен через кластер Hadoop [5,6]. HDFS реалізується двома службами: NameNode і DataNode. Реалізація HFAA сумісна з OrangeFS (вилкою PVFS) і порівнює його продуктивність по відношенню до інших підходів. Нарешті, в розділі 6 обговорюються споріднені роботи в нашому підході.

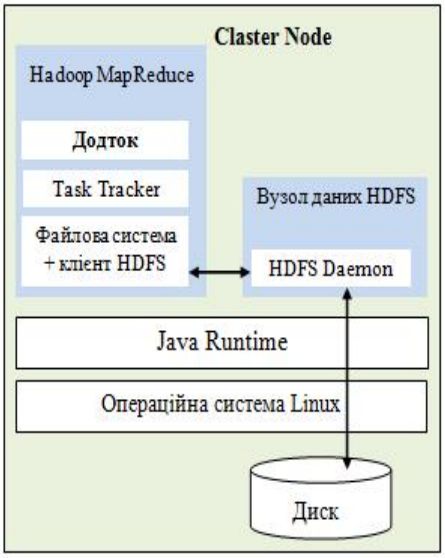

Рис. 1 – Кластерний вузол Hadoop

Іменований вузол NameNode відповідає за підтримання дерева каталогів HDFS, і являється централізованою службою в кластері, працюючому з єдиним вузлом. Клієнти звертаються до іменованого вузла для виконання звичайних операцій файлової системи, таких як відкриття, закриття, перейменовування і видалення. Цей вузол не зберігає самих даних HDFS, а скоріше підтримує відображення між іменем файлу HDFS, списком блоків у файлі і вузлом/вузлами даних, на якому/яких ці блоки зберігаються.

До того ж, до централізованого іменованого вузла всі інші вузли кластера обслуговують вузол даних. Кожен вузол даних зберігає блоки HDFS (64MB, фрагмент пам'яті одного логічного файла) від локальних або віддалених клієнтів. Кожен блок зберігається у вигляді окремого файлу в локальній файловій системі вузла, який використовує власну файлову систему, таку як ext4. Блоки створюються або знищуються на вузлі даних за запитом іменованого вузла, який перевіряє і обробляє запити від клієнтів. Хоча іменований вузол управляє простором імен, клієнти все ж безпосередньо спілкуються з вузлом даних для читання або запису даних на рівні блоків HDFS.

На рисунку 1 показана архітектура стандартного вузла Hadoop кластера, що використовується для обчислень і збереження. Двигун Map-Reduce (працюючий у віртуальній машині Java) виконує додаток користувача. Коли додаток зчитує або записує дані, запити передаються через клас Hadoop org.apache.hadoop.fs.FileSystem, який забезпечує стандартний інтерфейс для розподілених файлових систем, в тому числі HDFS за замовчуванням. Клієнт HDFS є все ж відповідальним за отримання даних від розподіленої файлової системи через з'єднання з DataNode з необхідним блоком. У загальному випадку DataNode працює на тому ж вузлі, а тому не потрібно ніякого зовнішнього мережевого трафіку. DataNode, також працюючи у віртуальній машині Java, доступається до даних, що зберігаються на локальному диску, використовуючи звичайний файл функцій вводу/виводу.

Масштабовані файлові системи

Існує багато розподілених файлових систем, створених для масштабування через великій комп'ютерний кластер. Деякі з них є відкритим вихідним кодом, у тому числі Ceph [15], Lustre [16], і PVFS [17], в той час як інші є власними, в тому числі GPFS [18], файлової системи Google [19,20], і PanFS [21].

Функція	HDFS	Lustre	Ceph	PVES
Простір імен	Централізований	Централізований	Розподілений	Розподілений
Витримки помилок	Звернення до файла та авто-оновлення	Апаратна залежність	Звернення до файла та авто-оновлення	Апаратна залежність
Симантики запису	Єдиний запис (забезпечується підтримка додавання інформації до файла	Запис і перезапис будь-де з участю більшості POSIX-семантик		
Доступність	Звичайні API Hadoop або драйвер ядра (відмінний від POSIX)	Звичайний (Lustre/ Ceph/PVFS) API або драйвер ядра, забезпечений POSIX-подібною функціональністю		
Розвернута модель	Вузли, що використовуються для обчислень та збереження	Розподвлені вузли, що використовуються для обчислень та збереження (призначені служби)		

Таблиця 1 – Порівняння HDFS та інших розподілених файлових систем

У таблиці 1 наводиться порівняння системи HDFS та інших розподілених файлових систем з відкритим вихідним кодом, включаючи Ceph, Lustre, і PVFS. Основні відмінності включають обробку файлових метаданих в централізованому або розподіленому режимі, реалізація відмовостійкості по причині апаратного RAID (нижче рівня файлової системи) або на базі реплікації програмного забезпечення і накладень без POSIX обмежень, таких як файл-записи -once (тільки один раз). У широкому сенсі розуміння HDFS була розроблена спеціально для підтримки одного виду програмної моделі – MapReduce і, таким чином, має спрощений дизайн, сильно оптимізований в напрямку до потокового доступу. В протилежність їй, інші паралельні розподілені файлові системи були розроблені для високопродуктивних обчислювальних середовищ, в основу яких положені кілька моделей програмування, таких як MPI, PVM, OpenMP і, таким чином, вони повинні підтримувати більш широкий спектр можливостей.

4. Агностик API файлової системи Hadoop (HFAA API)

Агностик API для файлової системи Hadoop (HFAA) забезпечує простий спосіб взаємодії Hadoop з файловими системами, відмінними HDFS. Вона орієнтована на розробників цих альтернативних файлових систем, тобто, тих, хто має уявлення про свою розподілену файлову систему, але не є обізнаними у внутрішній роботі системи Hadoop. HFAA усуває необхідність в розширеннях користувача для реалізації Hadoop в Java і забезпечує зв'язок з будь-якою файловою системою через TCP сокети. Архітектура HFAA в загальному добре описана і для обговорення проектування розподілені і паралельні файлові системи, відмінні від співрідних з HDFS (наприклад, PVFS, Ceph, або Lustre) будемо називати комбінованим іменем – нові розподілені файлові системи (NewDFS).

Цей API призначений для розгортання в типовому середовищі Hadoop, де кожен вузол кластера відповідає за обчислення (через Hadoop MapReduce) і зберігання (через HDFS). Коли HDFS замінюється всім необхідним "NewDFS", кожен вузол буде як і раніше відповідати за обчислення і збереження. HDFS daemon буде відключена, а замість неї буде налаштована NewDFS daemon.

Щоб включити Hadoop MapReduce для зв'язку з NewDFS daemon, архітектура HFAA додає дві програмні компоненти для системи: клієнт HFAA і сервер HFAA, – як показано на рисунку 2. Клієнт HFAA інтегрується зі стандартною Hadoop MapReduce framework і перехоплює запити читання/запису, які зазвичай надходять в HDFS. Потім клієнт направляє запити через мережеві сокети на сервер HFAA, які, як правило, будуть працювати на тому ж вузлі, але може бути віддаленй. Сервер HFAA відповідає за взаємодію з NewDFS, отримуючи мережеві запити від клієнта HFAA для виконання операції файлової системи (створення, відкриття, видалення тощо) і відповідає за результати даних/метаданих з NewDFS.

Клієнт HFAA створений в Java інтегрований з Hadoop framework і може бути повторно використаний з будь-яким NewDFS. На відміну від цього, сервер HFAA написаний на будь-якій

іншій мові, на якій реалізований NewDFS, відповідно, і повинен бути переписаний для кожної нової з'єднаної файлової системи. Ми можемо побачити, що цей компонент наданий розробниками NewDFS, які не мають ніякої потреби в детальному знанні внутрішньої роботи Hadoop. Детальна інформація проектування та експлуатації клієнта і сервера HFAA описані нижче.

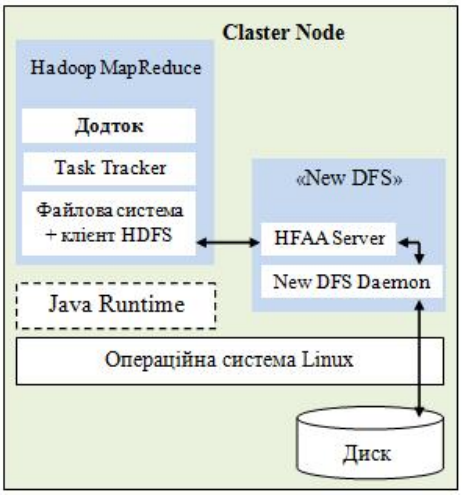

Рис. 2 – Вузол кластера HFAA

Інтеграція клієнта з Hadoop

Реалізація Hadoop в Java включає абстрактний клас під назвою org.apache.hadoop.fs.FileSystem. Кілька файлових систем розширюють і реалізують цей абстрактний клас, у тому числі HDFS (файлову систему за замовчуванням), Amazon S3, і Kosmos File System. Клієнт HFAA інтегрується з Hadoop шляхом реалізації цього класу файлової системі, по суті, змінюючи інтерфейс HDFS з еквівалентною функціональністю на інтерфейс для взаємодії з будь-якими NewDFS через TCP сокети.

Функція	Опис
Відкрити (Open)	Відкрити файл для читання.
Створити (Create)	Створити файл для запису.
Додати (Append)	Додати дані в існуючий файл.
Переіменувати (Rename)	Змінити ім'я файла або/чи перемістити файл в іншу директорію.
Видалити (Delete)	Видалити файл/директорію разо з їх інформацією.
Статус виводу списку (ListStatus)	Вивід списку файлів поточної директорії.
Створити директорію (MkDir)	Створити директорію за вказаним шляхом в файловій системі. Наддиректорії мусять уже бути створені і існувати на цей

	момент в файловій системі.
Статус виводу атрибутів (GetFileStatus)	Виявляє метадані (атрибути: об'єм, локалізація блоку, спосіб відповіді, власник, …) для вказаного файла чи директорії.
Запис (Write)	Операція запису реалізована за допомогою нового класу OutputStream.
Читання (Read)	Операція читання реалізована за допомогою нового класу OutputStream.

Таблиця 2 – Операції HFAA

Ключова функціональність HFAA реалізує включення, подані в таблиці 2. Для згенерації протоколу простий запит команди і відповідні типи кодів та їх дані/метадані обмінюються між клієнтом HFAA і сервером HFAA у вигляді рядкового формату. Групи запитів і відповідей є гнучкими і простими до організації роботи, тому що більшість мов програмування пропонують бібліотеки для роботи з рядками. Повний технічний опис протоколу HFAA знаходиться в [22].

Інтеграція сервера з NewDFS

Сервер HFAA функціонально взаємодіє з NewDFS. Він може приймати мережеві запити від клієнта HFAA для виконання операцій файловою системою (наприклад, створювати, відкривати, видаляти і т.д.) і відкликається на відповідні дані/метадані, отримані від NewDFS. У цій архітектурі функції інтерфейсу між клієнтом HFAA і сервером HFAA стандартизовані, але функції інтерфейсу між HFAA сервером і NewDFS daemon залишаються для реалізатора. Таким чином, сервер HFAA може бути інтегрований в самій NewDFS daemon, або залишений як автономна программа.

Незалежно від того, чи є він автономний чи інтегрований з демона NewDFS, базова структура сервера HFAA полягає в наступному. Сервер HFAA прослуховує тісно переплетені вхідні запити TCP, проглядаючи кожен запит як незалежну нитку або завершальний процес. Сервер HFAA повинен бути в змозі реагувати на численні запити одночасно, оскільки типовий Hadoop вузол паралельно обробляє декілька завдань MapReduce, і тому що кожне завданя має принаймні один файл даних і кілька файлів метаданих (відстеження статусу виконання завдання), що відкриваються в будь-який момент часу. У кожному процесі є багато обробників запитів для кожної функції API Hadoop (наприклад, getFileStatus(), listStatus()). Обробники запитів обслуговують запити повнісю до завершення, а потім виходять з них.

6. Роботи, пов'язані з даною тематикою

Інтеграції функціональності MapReduce з Hadoop з файловими системами, відмінними від HDFS в останні роки був проявлений значний інтерес. Така інтеграція типічно наслідує один з двох можливих підходів: розширення вбудованого абстрактного класу файлової системи для Hadoop зі вставкою коду для підтримки бажаної розподіленої файлової системи, або з використанням POSIX-драйвера щоб експонувати розподілену файлову систему в базову операційну систему, а потім безпосередньо використовувати цей ресурс з середини Hadoop.

Приклади альтернативних розподілених файлових систем, які були успішно використані з Hadoop MapReduce включають: Amazon S3 [9], Ceph [11], CloudStore/KosmosFS [10], GPFS [12], PVFS [14], і Lustre [13]. Багато з цих файлових систем можуть бути використані в інтеграції одного з наведених підходів: підходу інтеграції POSIX-драйвера або підходу вставки коду для підвищення продуктивності. Ключова різниця між усіма цими підходами і підходу, обраного для HFAA, полягає в тому, що HFAA був розроблений для підтримки постійності в реалізації Hadoop,

і було висунуто спрощений набір реалізованих відповідальних моментів в напрямку розподіленої файлової системи. На відміну від цього, всі попередні інтегровані проекти включали в себе свою власну відмінність, унікальний набір модифікацій і виправлень в базі коду Hadoop. Для отримання детальної додаткової інформації з проектування HFAA та протоколу необхідно звертатися до роботи [22].

Обговорення результатів

Архітектура HFAA клієнта і сервера, описані вище, були реалізовані і протестовані з Hadoop 0.20.204.0 і OrangeFS 2.8.4, – розподіленою файловою системою типу PVFS. Клієнт HFAA був розроблений в Java і скомпільований в Hadoop framework. HFAA сервер був написаний в C, для того щоб надати перевагу існуючому коду PVFS. Для зручності реалізації сервер HFAA не був безпосередньо інтегрований в daemon PVFS збереження. Швидше, він був реалізований як автономний процесс,схожий за стилем на існуючу програму AdminTools, що поставляється з новою файловою системою. Ці утиліти пропонують команди в стилі POSIX (наприклад, ls, touch) для підтримки та адміністрування PVFS.

Для багатогранної оцінки використовувалися чотири однорідні сервери. Кожен сервер складався з процесора Intel Xeon X3430, оперативної пам'яті4 ГБ, і жорсткий диск SATA 500 Гб. Для зберігання даних HDFS було виділено розділ на зовнішньому краю диска в 100 ГБ (тобто, найшвидша область) або дані PVFS. Використання лише невеликої області диска виявляє різницю пропускної здатності при доступі між найшвидшим і самим повільним секторами враховуючи фізичне розташування диска і зводиться до мінімуму – нижче 8% в експериментах. Кожен сервер був налаштований з Ubuntu 10.10 Linux-дистрибутива і програмного стека (Hadoop + розподілена файлова система), яка змінюється в експерименті. Всі вузли були пов'язані з допомогою свіча Gigabit Ethernet в закритій мережі.

В таблиці 3 подано три різні конфігурації програмного забезпечення кластера, що використовувався для експериментів: Hadoop з HDFS, Hadoop з PVFS і Hadoop з PVFS що використовує HFAA. Для збереження однаковості експериментів, у всіх конфігураціях використовувався один "старший" сервер для планування завдань, і три "допоміжні" сервери – для обчислення і збереження файлів даних. Розташування файлових метаданих змінювалося між експериментами.

Перша конфігурація, позначена Hadoop + HDFS – це стандартна Hadoop архітектура, де головний вузол стартує централізований JobTracker та послуги іменованого вузла (NameNode – для управління завданнями і простором імен файлової системи) і кожен підлеглий вузол обумовлює працювати власний примірник TaskTracker і послуги вузла даних (Data-Node – для обчислень і розподіленого збереження).

У другій конфігурації, що названа Hadoop + PVFS, HDFS була видалена і замінена PVFS. Драйвер POSIX-клієнта, що забезпечувався PVFS, використовується для монтування розподіленої файлової системи на кожному вузлі, і Hadoop налаштований на використання "локальної" директорії файлової системи безпосередньо через інтерфейс RawLocalFileSystem. У цій конфігурації Hadoop обмежується інтерфейсом POSIX-драйвера, і не може отримати інформаційної локальності з наголошеної файлової системи.

Архітектура PVFS розроблена з 3-х компонентів: сервери метаданих (зберігають метадані), сервери вводу-виводу (для зберігання даних), і клієнти. Для узгодженості з конфігурацією HDFS, головний вузол в конфігурації Hadoop + PVFS використовується тільки для запуску сервера метаданих, а всі інші підпорядковані вузли працюють як сервер метаданих і сервер вводу/виводу. Таким чином, як і в конфігурації HDFS, тільки 3 підпорядковані вузли зберігають дані файлу. Однак, на відміну від HDFS, сервер(и) метаданих (еквівалент іменованих вузлів) розподіляється по всьому суцільному кластеру.

В третій конфігурації Hadoop + HFAA + PVFS, змінюється інтерфейс між Hadoop і PVFS. POSIX-драйвер і PVFS-клієнт видаляються, і заміняються новою клієнт-серверною архітектурою HFAA як це буде обговорюватися нижче. Як і в попередній конфігурації, кожен вузол виконує сервер метаданих, але тільки підпорядковані вузли запускають сервер вводу/виводу з метою відтворення оригінальної HDFS-архітектури.

Вузол	Hadoop+HDFS	Hadoop+PVFS	Hadoop+HFAA+PVFS
Головний (1)	H: JobTracker D: ІменованийВузол (метадані)	H: JobTracker P: POSIX-драйвер P: Клієнт P: СерверМетаданих (метадані)	H: JobTracker A: HFAA-клієнт A: HFAA-сервер P: СерверМетаданих (метадані)
Підлеглий (3)	H: TaskTracker D: ВузолДаних(дані)	H: TaskTracker P: POSIX-драйвер P: Клієнт P: СерверМетаданих (метадані) P: СерверВ-В(дані)	H: TaskTracker A: HFAA-клієнт A: HFAA-сервер P: СерверМетаданих (метадані) P: СерверВ-В(дані)

Таблиця 3: Позначення конфігурації кластера для програмного забезпечення: H = Hadoop MapReduce, D = HDFS, P = PVFS, A = HFAA

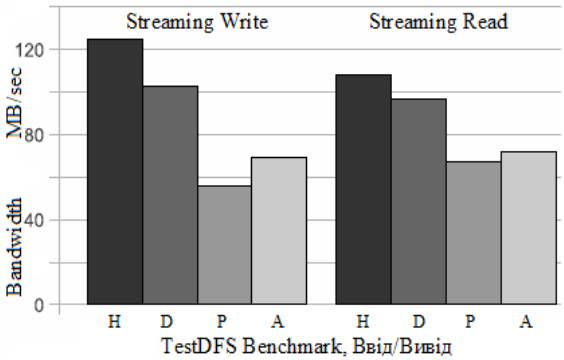

Рис. 3 – Пропускна здатність на запис і читання при доступі до файла для варіантів, поданих в таблиці 3

Кожна із трьох кластерних конфігурацій багатогранно протестовані з додатком TestDFSIO для MapReduce, який поставляється з Hadoop framework. Цей тест виконує паралельні операції потокового читання або потокового запису для багатьох файлів в рамках кластера. Кожен окремий файл зчитується або записується одним завданням відображення програми TestDFSIO, і кожен підлеглий кластер може виконувати кілька завдань відображення одночасно. Так як кожне завдання відображення виконується в межах кластера, одиничні завдання зменшують агрегативність завдань і статистику звітів продуктивності виконання. У цих експериментах TaskTracker на кожному підлеглому вузлі виконує єдине завдання відображення за допомогою TestDFSIO: читання або запис єдиного файла на 5 Гб в глобальній файловій системі, і в цілому 15 Гб доступних даних в сукупності по всьому кластеру. Слід звернути увагу, що через те що реплікація природньо не підтримувалася PVFS, воная була відключена у всіх конфігураціях.

На малюнку 3 подані результати TestDFSIO для трьох кластерних конфігурацій для робочих операцій читання і запису. Ці результати порівнюються з результатом обробки сировинного диска повністю поза середовищем Hadoop, який був виміряний за допомогою утиліти dd, зконфігурованої для роботи з потоковим доступом. При стандартній конфігурації Hadoop + HDFS досягається 82% від середньої пропускної здатності для запису і 87,3% від середньої пропускної здатності для читання порівняно з пропускною здатністю для необробленого диска. На відміну від цього, при конфігурації Hadoop + PVFS досягається тільки 46% від середньої пропускної здатності для запису і 60,3% від середньої пропускної здатності для читання в порівнянні з пропускною здатністю для необробленого диска. Нарешті, конфігурація Hadoop + HFAA + PVFS дозволяє досягти 54,9% від середньої пропускної здатності для запису і 64% від середньої пропускної здатності для читання в порівнянні з вихідною пропускною здатністю для необробленого диска, показуючи при цьому, як HFAA може зберегти оригінальну пропускну здатність файлової системи і зробити її доступною для Hadoop.

Одержані результати виявляють функціональні можливості інтерфейсу HFAA. Однак, використання цього API з PVFS не підходить, а також не перевершує рівні вводу/виводу HDFS, підтримується 67,2 % з пропускної здатності для запису HDFS і 74,1% з пропускної здатності для читання HDFS. Інтеграція PVFS з HFAA досягає ~23% приросту пропускної здатності для запису і

~7% приросту пропускної здатності для читання в порівнянні з використанням PVFS як локальної файлової системи через інтерфейс POSIX-ядра.

Висновки

Агностик API файлової системи Hadoop розширює Hadoop з простим протоколом зв'язку, що дозволяє інтегруватися їй з будь файлової системою, яка підтримує сокети TCP. Цей новий API усуває необхідність у налаштуванні реалізації Hadoop в Java, і замість цього переміщає функції відповідальності реалізації на саму файлову систему. Таким чином, розробники, бажаючі інтегрувати їх нову файлову систему з Hadoop не несуть відповідальності за розуміння деталей внутрішньої роботи в Hadoop.

Надаючи файловій системі агностик API майбутня робота може безпосередньо досліджувати компроміси продуктивності з використанням програмної моделі Map-Reduce з файловими системами, що підтримують операції записів в будь-якому місці, розподілених серверів метаданих та інших переваг. Крім того, HFAA дозволяє Hadoop інтегруватися більш легше в існуючих обчислювальних високопродуктивних середовищах, де заміщення здійснюється вже відомими розподіленими файловими системами.

1. Hadoop. http://hadoop.apache.org, 2012.
2. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In OSDI'04: Proceedings of the 6-th Symposium on Opearting Systems Design & Implementation, page 10, Berkeley, CA, USA, 2004.
3. Powered by Hadoop. http://wiki.apache.org/hadoop/PoweredBy, 2012.
4. S. Radia. HDFS federation - apache Hadoop india summit. http://www.slideshare.net/huguk/hdfs-federation-hadoop-summit, 2011.
5. HDFS (Hadoop distributed file system) architecture. http://hadoop.apache.org/common/docs/current/hdfs_design.html, 2012.
6. K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed file system. In Proceedings of the 2010 IEEE 26-th Symposium on Mass Storage Systems and Technologies (MSST), MSST '10, pages 1 – 10, Washington, DC, USA, 2010.
7. K. V. Shvachko. HDFS scalability: The limits to growth. In login: The Magazine of USENIX, volume 35, pages 6 – 16, Apr. 2010.
8. 8.K. V. Shvachko. Apache Hadoop: The scalability update. In login: The Magazine of USENIX, volume 36, pages 7 – 13, June 2011.
9. Amazon S3. http://wiki.apache.org/hadoop/AmazonS3, 2012.
10. kosmosfs - kosmos distributed filesystem. http://code.google.com/p/kosmosfs/, 2012.
11. C. Maltzahn, E. Molina-Estolano, A. Khurana, N. A. J., S. A. Brandt, and S. A. Weil. Ceph as a scalable alternative to the Hadoop distributed file system. In login: The Magazine of USENIX, volume 35, pages 38 – 49, Aug. 2010.
12. R. Ananthanarayanan, K. Gupta, P. Pandey, H. Pucha, P. Sarkar, M. Shah, and R. Tewari. Cloud analytics: do we really need to reinvent the storage stack? In Proceedings of the 2009 conference on Hot topics in cloud computing, HotCloud'09, Berkeley, CA, USA, 2009.
13. Using Lustre with Apache Hadoop. http: //wiki.lustre.org/images/1/1b/Hadoop_wp_v0.4.2.pdf, Jan. 2010.
14. W. Tantisiriroj, S. W. Son, S. Patil, S. J. Lang, G. Gibson, and R. B. Ross. On the duality of data-intensive file system design: reconciling HDFS and PVFS. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, pages 67:1 – 67:12, New York, NY, USA, 2011.
15. S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: a scalable, high-performance distributed file system. In Proceedings of the 7th symposium on Operating systems design and implementation, OSDI '06, pages 307 – 320, Berkeley, CA, USA, 2006.
16. Lustre file system. http://www.lustre.org, 2012.
17. Parallel virtual file system, version 2. http://www.pvfs.org, 2012.
18. F. Schmuck and R. Haskin. Gpfs: A shared-disk file system for large computing clusters. In Proceedings of the 1-st USENIX Conference on File and Storage Technologies, FAST '02, Berkeley, CA, USA, 2002.
19. S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 29 – 43, New York, NY, USA, 2003.
20. M. K. McKusick and S. Quinlan. GFS: Evolution on fast-forward. Queue, 7:10:10 – 10:20, August 2009.
21. B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the panasas parallel file system. In Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST'08, pages 2:1 – 2:17, Berkeley, CA, USA, 2008.
22. A. J. Yee. Sharing the love: A generic socket API for Hadoop mapreduce. Master's thesis, University of the Pacific, 2011.