УДК 004.657

© 2016 г. **Ю.А. Григорьев**, д-р техн. наук,
А.И. Устимов

(Московский государственный технический университет им. Н.Э. Баумана)

СРАВНЕНИЕ ВРЕМЕНИ ВЫПОЛНЕНИЯ ЗАПРОСА К ХРАНИЛИЩУ ДАННЫХ В СРЕДЕ MAPREDUCE/HADOOP И СУБД MYSQL

В облачной архитектуре реализован метод доступа к хранилищу данных (ХД) в среде MapReduce/Hadoop, который в зарубежных публикациях получил название Multi-Fragment-Replication Join (MFRJ). По результатам натурных экспериментов выполнено сравнение времени выполнения запроса к ХД с использованием метода MFRJ и в среде СУБД MySQL. Показано, что при увеличении объема таблицы фактов метод MFRJ имеет преимущества.

Ключевые слова: MapReduce/Hadoop, СУБД MySQL, хранилище данных, время выполнения запроса.

DOI: 10.22250/isu.2016.49.3-12

Введение

«Большие Данные (Big Data) – это уже устоявшаяся сфера технологий, даже несмотря на относительно молодой ее возраст, получившая распространение во многих сферах бизнеса и играющая немаловажную роль в развитии компаний ... За период с 2012 по 2014 год количество данных, ежемесячно передаваемых мобильными сетями, выросло на 81%. По оценкам Cisco в 2014 году объем мобильного трафика составил 2,5 эксабайтов (1 эксабайт = 10^{18} байтов) в месяц, а уже в 2019 году он будет равен 24,3 эксабайтам» [1].

Большинство компаний используют "Большие Данные" в сфере клиентского сервиса (повышение качества обслуживания клиентов – 53%), второе по популярности направление – операционная эффективность (повышение качества продукции и снижение затрат – 40%).

В настоящее время при использовании Big Data востребованы следующие технологии [1]:

на основе реляционных баз данных SQL – 42% (платформы in-memory, columnar [2] и др.);

использующие базы данных NoSQL – 44% (собственно БД NoSQL, платформа MapReduce/Hadoop и др.).

Возникает задача сравнения этих двух технологий. В данной статье выполнено сравнение метода доступа к хранилищу данных (ХД) по технологии MapRe-

duce/Hadoop и метода, применяемого в реляционных базах данных (на примере СУБД MySQL).

Постановка задачи

Тестируемое хранилище данных, которое представлено на рис. 1, имеет три измерения (Месяцы, Города, Товары) и одну таблицу фактов (Объемы продаж). Это хранилище было реализовано в среде MapReduce/Hadoop (в дальнейшем просто Hadoop) и с использованием СУБД MySQL. К ХД выполнялся запрос: в Hadoop – при помощи задания MapReduce, а в MySQL – при помощи SQL-запроса (см. рис. 1).

Рис. 1. Постановка задачи.

Необходимо было сравнить время выполнения запросов T1 и T2. В Hadoop запрос может выполняться параллельно на нескольких узлах, по которым распределено хранилище данных. Но эта технология не поддерживает индексирование данных в стандартном исполнении. В MySQL, наоборот, – запрос выполняется на одном узле (как в большинстве непараллельных реляционных СУБД), но запрос оптимизируется и при его выполнении используются индексы по первичным ключам таблиц измерений (id) и внешним ключам таблицы фактов (Месяц, Город, Товар). Поэтому заранее предсказать результаты тестирования невозможно.

Метод MFRJ доступа к хранилищу данных в среде Hadoop

В [3] рассматриваются способы реализации соединения таблиц измерений и фактов, которые связаны по схеме «звезда». Они позволяют избежать перемещений записей таблицы фактов. Рассмотрим пример запроса к ХД для схемы с двумя измерениями (D1 и D2) и одной таблицей фактов (F):

```

SELECT D1.d11, D2.d21, F.m1
FROM D1 JOIN F ON (D1.d10 = F.fk1) JOIN D2 ON (D2.d20 = F.fk2)
WHERE CD1 AND CF1,
    
```

(1)

где CD1 и CF1 – некоторые дополнительные условия, накладываемые на измере-

ния и факты.

Ниже приведен алгоритм соединения с использованием метода Multi-Fragment-Replication Join (MFRJ) [2, 3].

Таблицы измерений продублированы по всем узлам системы Hadoop. Они хранятся по строкам. Таблица фактов делится на несколько групп (рис. 2): в 1-ю группу входят столбцы (колонки) внешних ключей измерений (fk_i), столбец каждого факта (m_i) образует отдельную группу (группы 2, ..., $k+1$). Все блоки групп таблицы фактов распределены Hadoop по узлам (автоматически) произвольным образом (система пытается делать это равномерно).

Рис. 2. Таблица фактов.

В соответствии с запросом типа (1) доступ к данным выполняется следующим образом.

Мар (в каждом узле):

1. Для измерений, которые участвуют в запросе, строятся хеш-индексы в оперативной памяти.

2. Читаются записи внешних ключей измерений таблицы фактов (группа 1), хранящиеся в узле (fk_i); для каждой строки (позиции) и каждого внешнего ключа проверяется наличие значения этого ключа в соответствующем хеш-индексе; при успешном сравнении в выходной поток помещается запись:

$$\langle \text{позиция1}, (vd_1, \dots, vd_n) \rangle, \quad (2)$$

где «позиция1» – номер строки в группе внешних ключей (нумерация сквозная по всем строкам группы 1); $(vd_1, \dots, vd_n)$ – список значений требуемых по условию запроса столбцов таблиц измерений (выбираются из хеш-индексов).

3. Читаются значения колонки i -го факта (группа $i+1$), хранящиеся в узле. Для каждого значения факта проверяется условие CF1 запроса, и это значение помещается в выходной поток:

$$\langle \text{позиция2i}, vm_i \rangle, \quad (3)$$

где «позиция2i» – номер строки в группе $i+1$ (нумерация сквозная по всем строкам группы $i+1$); vm_i – значение i -го факта.

Значения ключей «позиция1» и «позиция2i» соответствуют одной таблице фактов, но по условию блоки групп таблицы фактов распределены по узлам произвольно.

4. Далее пункт 3 повторяется для других колонок фактов, блоки которых

хранятся в данном узле ($i=1, \dots, k$).

Reduce:

1. Если в полученной функцией Reduce записи (после группирования по позиции) число элементов в области значения равно $n+k$ (n – число измерений, k – число фактов) и она удовлетворяет условию запроса (проверяются заданные отношения между столбцами), то эта запись помещается в выходной поток как строка результирующей таблицы:

$$\langle \text{позиция}, (vd_1, \dots, vd_n, vm_1, \dots, vm_k) \rangle. \quad (4)$$

Реализация метода MFRJ

Метод MFRJ реализован в облачной архитектуре. Хранилище данных на основе Hadoop развернуто на виртуальных машинах с операционной системой Ubuntu. Для хостинга выбран облачный провайдер Digital Ocean [4].

В качестве экспериментального макета используется кластер до четырех узлов (см. рис. 1), в котором одна из машин является управляющей (NameNode), а на остальных машинах хранятся данные (DataNode). Архитектура системы такова, что можно с легкостью добавлять и убирать узлы кластера. Репликация настроена таким образом, что хранилище данных равномерно распределено между узлами DataNode.

Фаза Map (рис. 3). На каждом из N узлов DataNode одновременно запускаются L экземпляров функции Map ($L = 1$ или 2), которые выполняются параллельно и читают из файловой системы HDFS записи (объекты) таблиц измерений и фактов в виде пары <ключ, значение>. Принятые записи преобразуются в новые записи <ключ1, значение1> (2), (3) и выводятся в буфер памяти.

Рис. 3. Реализация метода MFRJ.

Каждый экземпляр Map обрабатывает записи одного входного файла. Размер обрабатываемого раздела не превышает размер одного split (по умолчанию

его размер равен размеру блока $V_S = V_B = 128$ Мбайтов).

Для каждого экземпляра Map выделяется буфер памяти объемом $Q_B = 100$ Мбайтов. Когда объем заполнения буфера превышает определенный порог, запускается отдельный поток (нить), который выполняется в фоновом режиме. Он разбивает буфер на фрагменты и сортирует каждый фрагмент по ключу (in-memory sort). Число фрагментов в буфере равно числу разделов s (заданий Reduce), указанному в задании (Job). Запись принадлежит i -му фрагменту буфера (т.е. i -му разделу), если $h(\text{ключ}1) = i$, где h – некоторая хеш-функция. После этого каждый отсортированный фрагмент буфера выводится на диск как часть файла (spill to disk). Таким образом, общее число отсортированных фрагментов, которые будут сохранены на диске в процессе выполнения одной функции Map узла, будет равно $r \cdot s$, где r – число сохраненных файлов.

После того как функция Map обработает все входные записи своего split, в узле запускается специальная процедура. Она сортирует записи $\langle \text{ключ}1, \text{значение}1 \rangle$ i -го раздела, объединяет их в один файл (merge on disk) и сохраняет на локальном диске (см. рис. 3).

Указанные выше действия выполняются параллельно для всех экземпляров Map, запущенных на данном узле. Сортировка необходима для выполнения операции группирования на фазе Reduce.

Фаза Shuffle. Как только экземпляр Map завершается, об этом оповещается центральный узел (NameNode, рис. 1). Одновременно с Map запускаются $s=2$ экземпляров задачи Reduce (s – общее число). Каждая j -я задача Reduce опрашивает узел NameNode, определяет завершенные Map и читает файлы j -х разделов в свою область, т.е. файлы разных узлов с одним номером раздела j будут переданы в узел, где выполняется j -я задача Reduce ($j = 1, \dots, s$). Важно подчеркнуть, что записи (2) и (3) с одним и тем же номером позиции обрабатываются одним экземпляром задачи Reduce. Это позволяет функции Reduce сформировать результирующие записи (4), которые соответствуют запросу (1) – см. VM3 на рис. 3.

Фазы Map и Shuffle выполняются параллельно, что подтверждено в [5].

Фаза Reduce. Полученные на фазе Shuffle файлы постепенно читаются и сортируются/сливаются в памяти. Образуется один отсортированный поток записей, который без промежуточного запоминания на диске передается на вход функции Reduce.

Перед тем как передать запись-группу на вход функция Reduce, система группирует входные записи по значению ключа, по которому выполняется сортировка. Далее функция Reduce обрабатывает каждую группу и помещает результат в выходной поток (4). Эти данные сохраняются в файловой системе HDFS.

После успешного выполнения задания результаты размещаются в нескольких физических файлах, образуя один распределенный файл с именем, который указал пользователь. Обычно не требуется объединять их в один файл, потому что часто полученные файлы используются в качестве входных для запуска следующего Hadoop-задания или в каком-либо другом распределенном приложении, которое может получать входные данные из нескольких файлов.

Анализ результатов натурных экспериментов

Описание макетов хранилищ данных на основе MySQL приведено в табл. 1.
Таблица 1

Обозначение	Количество узлов	Технические характеристики узла
MySQL512	1	512 Мб оперативной памяти. Одноядерный процессор. Дисковый накопитель SSD объемом 20 Гб.
MySQL4GB	1	4 Гб оперативной памяти. Двухядерный процессор. Дисковый накопитель SSD объемом 20 Гб.

Макеты хранилищ данных на основе Hadoop обладают одинаковой технической конфигурацией узлов, но различаются их количеством (табл. 2).

Таблица 2

Обозначение	Количество узлов	Технические характеристики узла
Hadoop2	2 узла, 1 из которых NameNode, второй DataNode.	4 Гб оперативной памяти. Двухядерный процессор. Дисковый накопитель SSD объемом 20 Гб.
Hadoop3	3 узла, 1 из которых NameNode, остальные DataNode.	
Hadoop4	4 узла, 1 из которых NameNode, остальные DataNode.	

Каждое хранилище обладает логически одинаковой структурой, представленной на рис. 1. Методика проведения экспериментов заключается в том, что в каждом из описанных выше макетов несколько раз (3 раза) выполняется один и тот же запрос, SQL-эквивалент которого представлен на рис. 1. Далее вычисляется среднее значение, которое и отображается на графике. Число записей в таблицах измерений равно: Месяцы – 12; Города – 100; Товары – 1000. Количество записей в таблице фактов хранилища данных увеличивается с шагом 1 млн. шт. (это соответствует приблизительно 50 Мбайтам).

Из рис. 4 видно, что зависимости времени выполнения запроса в СУБД MySQL практически линейные и совпадают для макетов MySQL512 и MySQL4GB.

Рис. 4. Зависимость времени выполнения запроса к ХД в СУБД MySQL от числа записей в таблице фактов.

Это объясняется тем, что здесь доминирующую роль играет подсистема ввода/вывода на диск, характеристики которой одинаковы для обоих макетов.

На рис. 5 показаны зависимости времени выполнения запроса в системе Hadoop для макетов Hadoop2, Hadoop3, Hadoop4.

Рис. 5. Зависимости времени выполнения запроса к ХД в системе Hadoop от числа записей в таблице фактов ($s = 2$).

Из рисунка видно, что при небольшом объеме данных в таблице фактов время выполнения запроса приблизительно одинаково для кластеров всех размеров. Это объясняется тем, что большая часть времени затрачивается на запуск виртуальной машины Java и создание контейнеров для выполнения задач Map и Reduce. При увеличении объема обрабатываемых данных наблюдаются достаточно большие скачки времени выполнения запроса, хотя при этом имеет место следующая тенденция: чем больше узлов в кластере, тем меньше время выполнения запроса.

Пилообразный характер зависимостей можно объяснить так. Здесь все натурные эксперименты проводились при числе задач Reduce (контейнеров), равным $s = 2$. Равномерность распределения записей по этим задачам в данном случае зависит от числа записей. При количестве записей 4, 6, 8, 10 млн. они более равномерно распределяются по двум задачам, а при 5, 7, 9 млн. – менее равномерно (по умолчанию номер задачи j определяется как остаток от деления значения ключа записи на s : $j = h(\text{ключ1}) = \text{ключ1}(\text{mod } s)$). Для макета Hadoop2 обе задачи (контейнера) запускаются на одном узле DataNode, они потребляют общие ресурсы, поэтому здесь неравномерность распределения сказывается меньше (см. рис. 5).

Для других макетов эти задачи запускаются на разных узлах DataNode, и здесь задействованы разные ресурсы, включая сетевые, поэтому неравномерность распределения сказывается больше.

Как видно из рис. 6, начиная с отметки 4 млн. записей в таблице фактов, время выполнения запроса в Hadoop меньше, чем в СУБД MySQL. При числе записей 10 млн. преимущество Hadoop4 становится существенным и достигает $(325 - 200)/325 = 38\%$.

На рис. 7 представлена зависимость времени выполнения запроса в макете ХД Hadoop4 от числа задач Reduce (т.е. от s). При увеличении s от 2 до 6 время уменьшается только на $(151 - 138)/151 = 9\%$. Это можно объяснить неравномер-

ностью распределения записей, помещаемых Map в выходной поток по задачам Reduce.

Рис. 6. Зависимость времени выполнения запроса к ХД в СУБД MySQL и в системе Hadoop от числа записей в таблице фактов.

Рис. 7. Зависимость времени выполнения запроса в макете ХД Hadoop4 от числа задач Reduce (т.е. от s), число записей в таблице фактов равно 10 млн.

Ниже представлены результаты исследования журналов log и работы монитора в среде Hadoop4 при выполнении запроса к ХД (10 млн. записей в таблице фактов, 2-е задачи Reduce).

На рис. 8 видно, что изначально из HDFS считывается 495 Мбайтов входных данных. Объем данных, записанных во временные файлы, составил 1945 Мбайтов. Такой большой объем объясняется выполнением сортировки-слияния промежуточных записей на фазах Map и Reduce. Объем считанных данных равен 1072 Мбайтам. Этот объем меньше объема записанных временных данных, что объясняется чтением некоторых данных из буфера ОП после сохранения их на диске. Из них 873 Мбайтов данных были переданы по сети (Shuffle) на вход задач Reduce (от 10 задач Map Hadoop использует спекулятивное выполнение Map).

На рис. 9 представлено, как изменяется количество задач Map и Reduce в процессе выполнения задания. Первоначально запускаются 5 (из 10) задач Map (они выполняются параллельно). По завершении первой из этих 5 задач (это около 77-й секунды) запускается задача Reduce. Через некоторое время количество задач Map уменьшается, и к 146-й секунде все задачи данной фазы завершены. На 227-й секунде задание MapReduce завершается полностью.

На рис. 10 – 12 показаны характеристики использования ресурсов кластера при выполнении задания MapReduce. Из графиков видно, что пик загрузки ресур-

са приходится примерно на отметку 123 секунды, когда в кластере активны и задачи Map, и задачи Reduce (см. рис. 9).

Рис. 8. Объем прочитанных и записанных данных в HDFS и временные файлы.

Рис. 9. Количество задач Map и Reduce и время их выполнения.

Рис. 10. Загрузка ЦП в процентах во время выполнения задания MapReduce.

Рис. 11. Использование оперативной памяти во время выполнения задания MapReduce.

Рис. 12. Сетевой трафик во время выполнения задания MapReduce.

Заключение

В облачной архитектуре (провайдер Digital Ocean) реализован метод MFRJ доступа к хранилищу данных в среде MapReduce/Hadoop. Выполнена серия натурных экспериментов с целью сравнения Hadoop с MySQL. При достаточно большом числе записей в таблице фактов макет Hadoop4 имеет заметное преимущество (38%), т.е. фрагментация данных по нескольким узлам в Hadoop и их параллельная обработка оказывают более существенное влияние на время выполнения запроса, чем его оптимизация и использование индексов в СУБД MySQL.

Небольшое уменьшение времени выполнения запроса (9%) при увеличении числа задач Reduce (контейнеров) объясняется неравномерностью распределения записей по этим задачам. Это свидетельствует о необходимости более тщательно подбора функции хеширования $h(\text{ключ1})$.

Осуществлено исследование журналов log и работы монитора в среде Hadoop4 при выполнении задания MapReduce. Оно подтверждает, что фазы Map и Reduce выполняются с некоторым временным сдвигом.

ЛИТЕРАТУРА

1. Аналитический обзор рынка BigData. [Электронный ресурс] <https://habrahabr.ru/company/moex/blog/256747/> Проверено 12.06.2016.
2. Григорьев Ю.А., Плутенко А.Д., Плужников В.Л., Ермаков Е.Ю., Цвященко Е.В., Пролетарская В.А. Теория и практика анализа параллельных систем баз данных. – Владивосток: Дальнаука, 2015.
3. Zhou G., Zhu Y., Wang G. Cache Conscious Star-Join in MapReduce Environments. Cloud-I '13 Proceedings of the 2nd International Workshop on Cloud Intelligence, 2013.
4. Digital Ocean. [Электронный ресурс]. [<https://www.digitalocean.com>] Проверено 12.06.2016.
5. Li B., Mazur E., Diao Y., McGregor A., Shenoy P.J. A platform for scalable one-pass analytics using MapReduce. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). – 2011. – P.985-996.

E-mail:

Григорьев Юрий Александрович – grigorev@bmstu.ru;

Устимов Артем Игоревич – artjom-ustimov@yandex.ru.