МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА

Факультет информатики и систем управления

Кафедра теоретической информатики и компьютерных технологий

Курсовой проект

по курсу «Компьютерные системы и сети»

«Фреймворк и файловая система для распределённой обработки больших данных в рамках концепции map-reduce»

Выполнил:
студент ИУ9-91
Выборнов А. И.

Руководитель:
Дубанов А. В.

Москва 2014

Содержание

Введение

бла блабла

1. Теоретическая часть

1.1. Map-reduce

Map-reduce — концепция, используемая для распределённых вычислений над *большими данными* в компьютерных кластерах. Модель представлена компанией Google в 2004 году.

Большие данные (Big data) — термин, характеризующий любой набор данных, который достаточно велик и сложен для традиционных методов обработки данных, а также набор технологий для обработки таких наборов данных.

Выполнение приложения согласно концепции map-reduce состоит из двух последовательных этапов, между которыми происходит группировка результатов:

- *map* — предварительная обработка входных данных,
- *reduce* — свёртка предварительно обработанных данных.

Несмотря на то, что прототипами этапов *map* и *reduce* послужили одноимённые функции, используемые в функциональном программировании, их семантика заметно отличается.

Базовым элементом в концепции map-reduce является структура (key, value) — (ключ, значение). Программирование представляет собой определение двух функций (квадратные скобки [ ] обозначают список):

- $map : (key, value) \to [(key, value)]$
- $reduce : (key, [value]) \to [(key, value)]$

Приведённое выше определение является достаточно абстрактным для прикладного применения. На практике достаточно следующего варианта выше приведённого определения:

- map : файл ввода $\to [(key, value)]$
- $reduce : (key, [value]) \to$ файл вывода

На рисунке 1 схематически изображена обработка данных с помощью map-reduce. Файл ввода разбивается на блоки, каждый из которых поступает на вход функции *map*. Функция map обрабатывает его и порождает список пар (ключ, значение). Множество полученных списков пар объединяется и, затем, группируется по ключу, получая для каждого ключа список всех ассоциированных с ним значений.

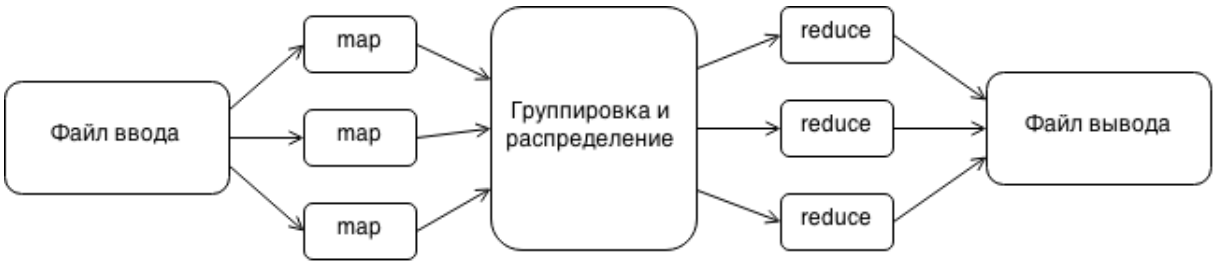

Рисунок 1 — Схематическое изображение обработки данных с помощью концепции map-reduce

Все ключи равномерно распределяются на блоки и передаются на вход стадии *reduce*. Стадия *reduce* преобразовывает ключ и список значений в блок, являющийся частью файла вывода.

Как видно из описания, концепция налагает ограничения на формат файлов ввода и вывода: данные должны быть коммуникативны, с некоторой точностью (обычно до строки).

1.1.1. Проблемы, которые решаются с помощью map-reduce

Концепция map-reduce, появившись в 2004 году, довольно быстро обрела популярность, так как накопилось множество проблем для которых не существовало универсального решения. Вот основные из них (в скобках указан подход к решению используемый в концепции map-reduce):

- вычисления превосходят возможности одной машины (кластеры из сотен и тысяч машин),
- данные не помещаются в памяти, необходимо обращаться к диску (последовательное чтение и запись намного эффективнее случайного доступа),
- большое количество узлов в кластере вызывает множество отказов (все узлы унифицированы, что упрощает восстановление работы после отказа),
- данные хранятся на множестве машин (данные обрабатываются на той же машине, на которой они хранятся),
- разработка низкоуровневых приложений для подобных систем — дорого и сложно (высокоуровневая модель программирования, универсальная и масшабируемая среда выполнения).

В рамках курсового проекта была предложена реализация концепции, дающая решение только части поставленных выше проблем, подробнее в главе 1.4.

1.1.2. Пример применения map-reduce

Разбор задачи на применение map-reduce

Задача: Есть граф пользователей некоторого ресурса, заданный в виде строчек: «пользователь - друг1 друг2 ...». Для каждой пары пользователей найти общих друзей.

Разберём задачу на следующих входных данных:

- A - B C D
- B - A C
- C - A B D
- D - A C

На стадии map преобразовываем пару (пользователь, друзья) в множество пар следующим образом:

- (A, B C D) -> (A B, B C D), (A C, B C D), (A D, B C D)
- (B, A C) -> (A B, A C), (B C, A C)
- (C, A B D) -> (A C, A B D), (B C, A B D), (C D, A B D)
- (D, A C) -> (A D, A C), (C D, A C)

Сливаем результаты полученные на стадии map, получаем список пар:

- (A B, [B C D, A C])
- (A C, [B C D, A B D])
- (A D, [B C D, A C])
- (B C, [A B D, A C])
- (C D, [A B D, A C])

На стадии reduce пересекаем с друг другом все элементы списка значений и получаем:

- (A B, C)
- (A C, B D)
- (A D, C)
- (B C, A)
- (C D, A)

1.2. Распределённая файловая система

Определение.

Обоснование необходимости

Архитектура распределённой файловой системы - картинка.

Описание того что есть на картинке

1.3. Распределённый map-reduce

Определение.

map-reduce удобная концепция, но ... ??

Архитектура распределённого map-reduce - картинка

Подробное описание картинки.

1.4. Решаемый класс задач

Необходимо обосновать существование и найти класс задач, для которого актуальна приведённая выше схема распределённого map-reduce.

Пусть каждый узел располагает $memory$ доступной оперативной памяти (здесь и далее единицей измерения памяти будет байт) и $storageSize$ свободного места на диске. Всего узлов $numNodes$. Данный map-reduce проектировался для решения задач с *большими данными*, поэтому в дальнейшем будем считать, что входные данные $>> memory$.

1.4.1. Ограничения на стадии map и reduce

На вход стадия map принимает данные с диска, а результат этой стадии равномерно (с точностью до пары (ключ, значение)) распределяется по оперативной памяти узлов. То есть входные данные $< storageSize * numNodes$, а результат $<< memory * numNodes$. Так как $storageSize >> memory$, получаем, что, в общем случае, стадия map должна сильно сокращать объём данных.

На вход стадия reduce принимает данные из оперативной памяти, которые $<< memory * numNodes$, результат этой стадии аккумулируется в оперативной памяти, то есть должен быть $<< memory$ для каждого узла, а затем записывается на диск. Получаем, что стадия reduce должна, как минимум, не увеличивать объём данных, поступивших на вход.

Рассмотрев полученные выше ограничения для стадий map и reduce, а также учитывая достаточно большие входные данные, можно увидеть, что распределённый map-reduce оптимально подходит для решения задач класса *информационный поиск* (*information retrieval*), что является одним из этапов решения задач *анализа данных* (*data mining*).

Информационный поиск — процесс поиска и получения информации как из структурированных, так и из неструктурированных данных. Обычно применяется в *анализе данных* для первичной обработки и сокращения объёма исходных данных.

1.4.2. Пример задачи класса информационный поиск

Исходные данные хранятся в виде текстового файла в n строк, в котором каждая строка соответствует строке в реляционной таблице $table$ с m столбцов ($f_1 ... f_m, m > 2$), каждое значение записано через пробел. Необходимо получить результат выполнения SQL запроса:

SELECT $f_1, ..., f_m$
FROM $table$
WHERE $f_1 > const$
GROUP BY f_1

1.4.3. Решение задачи с помощью фреймворка map-reduce

Функция map определяется следующим образом: для каждой строки проверяется условие **WHERE** и все строки, удолетворяющие условию, составляют результат в виде пар: $(f_1, f_2, ..., f_m)$. Количество строк, удолетворяющих условию **WHERE**, обозначим как n'. Функция reduce возвращает то, что получила на вход.

Стадия map получает $O(nm)$ данных и, по выполнении, выдаёт $O(n'm)$ данных, которые преобразуются и без изменений проходят стадию reduce. Сложность стадии map — $O(nm)$, преобразования между стадиями — за $O(n'm + n'ms_{net})$ (s_{net} — стоимость передачи данных между узлами), стадии reduce — $O(n')$.

1.4.4. Решение задачи на одной машине

Решение задачи одной машине без применения фреймворка состоит из двух стадий:

- получить из входного файла все строчки, удолетворяющие **WHERE** — сложность $O(nm)$ (по памяти $O(n'm)$)
- сгруппировать полученный результат по ключу f_1 — сложность $O(n'ln(n'))$, если применить для аггрегации быструю сортировку (по памяти $O(n'm)$), сложность $O(n')$, если применить для аггрегации сортировку подсчётом (по памяти $2O(n'm)$).

В случае когда n' и m определены таким образом, что получишиеся данные больше $memory$, возникают проблемы: необходимо сохранять на диск промежуточные результаты, полученные после первой стадии, и сортировать полученный результат на диске, что достаточно медленно. В наиболее быстром варианте (результат первой стадии по частям сортируется и сохраняется файлах, а затем эти файлы

сливаются) получается сложность первой стадии — $O(nm + n'ln(n') + n's_{hdd})$, второй стадии $O(n'ms_{hdd})$ (s_{hdd} — стоимость обращения к диску).

1.4.5. Выводы

Суммарная сложность решения с помощью map-reduce — $O(nm + n'm + n'ms_{net} + n')$, с помощью описанного выше способа решения на одной машине — $O(nm + n'ln(n') + n'ms_{hdd} + n'ms_{hdd})$. Для простоты положим, что $s_{hdd} \approx s_{net}$. Тогда можно увидеть, что принципиальное отличие в сложности заключается только в $O(n'ms_{hdd})$ для выполнения на одной машине, что соответсвует дополнительному слиянию файлов на диске.

На данном примере хорошо видно, что в случае выполнения на одной машине приходится кэшировать на диск то, что происходит в оперативной памяти нескольких нод в случае map-reduce. Как следствие — нужно вручную писать функционал, который реализован в фреймворке. Так же не стоит забывать, что время выполнения на одной машине существенно увеличивается не только за счёт слияния файлов, но и за счёт последовательного выполнения всех действий, которые происходят параллельно в случае map-reduce.

В итоге видно, что предложенная схема реализации концепции map-reduce позволяет решать задачи класса информационный поиск.

2. Объекты и методы

Характеристики программного обеспечения:

- Операционная система — Ubuntu 14.04.1 LTS 64-bit.
- IDE — Syblime Text 2.
- Язык программирования — Python 2.7.3.

Характеристики оборудования:

- Процессор — Intel Core i7-3770k 3.5Ghz×8.
- Оперативная память — 16Gb DDR3.
- Видеокарта — ATI Radeon 7860.

3. Реализация

3.1. Используемые технологии

При реализации ... были использованы готовые технологиии: ...

3.1.1. Коммуникация по сети

что такое
зачем нужно
почему zmq

3.1.2. Сериализация

что такое
зачем нужно
примеры, тесты
почему выбрали marshal
Сериализация в человекочитаемый формат
что такое и чем отличается от сериализации
json и xml определение и зачем нужно

3.2. Работа с большими данными

какие есть проблемы - описать все нижни пункты

3.2.1. python generator

Рассказ про python generator
и применение

3.2.2. split

обзор функции split

3.2.3. dfs

проблемы в dfs ?? доопределиться по ходу написания
как эти проблемы решаются в фс

3.2.4. map-reduce

проблемы в map-reduce ??
распределение ключей
как эти проблемы решаются в map-reduce

3.3. Взаимодействие между узлами

Описание реализации взаимодействия между различными узлами сети. класс nodesmanager и примеры его использования

3.4. Интерфейс

есть dfs, есть mr

3.4.1. dfs

перевести

Distributed file system is required to map-reduce framework.

On each node, you should run *dfsnode.py* with two arguments - port and storagepath. Like this:

python dfsnode.py -p 5556 -s /home/username/storage

Then you should fill *config.json* with information about nodes. Now you can use *dfs.py*. Samples of use dfs.py:

python dfs.py -ls /user/ python dfs.py -mkdir /user/username/userdatafolder python dfs.py -put ./test.in /user/username/userdatafolder/testfile python dfs.py -get /user/username/userdatafolder/testfile python dfs.py -rm /user/username

3.4.2. mr

написать аналогично dfs

4. Тестирование

5. Заключение

Описать успех или не успех тестирования.

Описать проблемы.

Описать удачи.

Список литературы

[1] гугловая статья про mr от 2006

лекции шад http://www.slideshare.net/yandex/mapreduce-12321523

http://www.quora.com/What-is-the-most-efficient-way-to-serialize-in-Python

http://www.codeinstinct.pro/2012/08/hadoop-design.html

https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/

[2] SMILES — A Simplified Chemical Language // Daylight Chemical Information Systems, Inc: URL: http://www.daylight.com/dayhtml/doc/theory/theory.SMILES.html

[3] Atomic Coordinate Entry Format Description // Penn State University: URL: http://www.wwpdb.org/documentation/format33/v3.3.html

[4] Periodic Table Datan Files // Protein Data Bank: URL: http://php.scripts.psu.edu/djh300/cmpsc221/p3s11-pt-data.htm

[5] Three.js — javascript 3D library // Three.js: URL: http://mrdoob.github.io/three.js/

[6] File: Tubby-1c8z-pymol.png // Wikipedia: URL: http://en.wikipedia.org/wiki/File:Tubby-1c8z-pymol.png

[7] GLmol - Molecular Viewer on WebGL/Javascript // GLmol: URL: http://webglmol.sourceforge.jp/index-en.html