Big Data’13

Лекция II: обработка данных при помощи Map-Reduce

Дмитрий Барашев
bigdata@barashev.net

Computer Science Center

28 февраля 2013

Сегодня в программе

Map-Reduce с точки зрения программиста

Некоторые задачи

- Расширение модели

Map-Reduce на коленке

Что такое Map-Reduce

- Модель программирования для параллельнной распределённой обработки больших объёмов данных
- Среда выполнения программ, написанных с использованием этой модели
- В промышленном программировании появилась благодаря Google
- Сейчас реализации MR используются многими компаниями для обработки логов, лайков, нахождения схожих объектов и т.п.

Сегодня в программе

Map-Reduce с точки зрения программиста

Некоторые задачи

- Расширение модели

Map-Reduce на коленке

Логическая модель

- Какие-то исходные данные – веб-страницы, логи, Википедия – разбитые на фрагменты (*shards*)
- Каждый фрагмент может содержать некоторое количество документов
- Три стадии: *Map* (маппинг, разбиение), *Shuffle* (сортировка), *Reduce* (сборка, свертка)

Логическая модель: Map

- Функции маппинга (*map function*) пережевывают входной фрагмент и выплевывают список пар ключ-значение
- $map_i : (doc_i, data_i) \to [(key_{ij}, value_{ij})]$
- Процессы map_i могут выполняться параллельно

Логическая модель: Shuffle

- Он называется по-английски shuffle, по-русски сортировкой, а на самом деле делает группировку
- Собирает вместе значения с одинаковыми ключами
- Shuffle:

$$[(key_{ij}, value_{ij})] \to \{R_k\} : \quad \begin{array}{l} key_{ij} = k \\ \forall (key_{ij}, value_{ij}) \in R_k \end{array}$$

- Получаются задачи для функций свертки

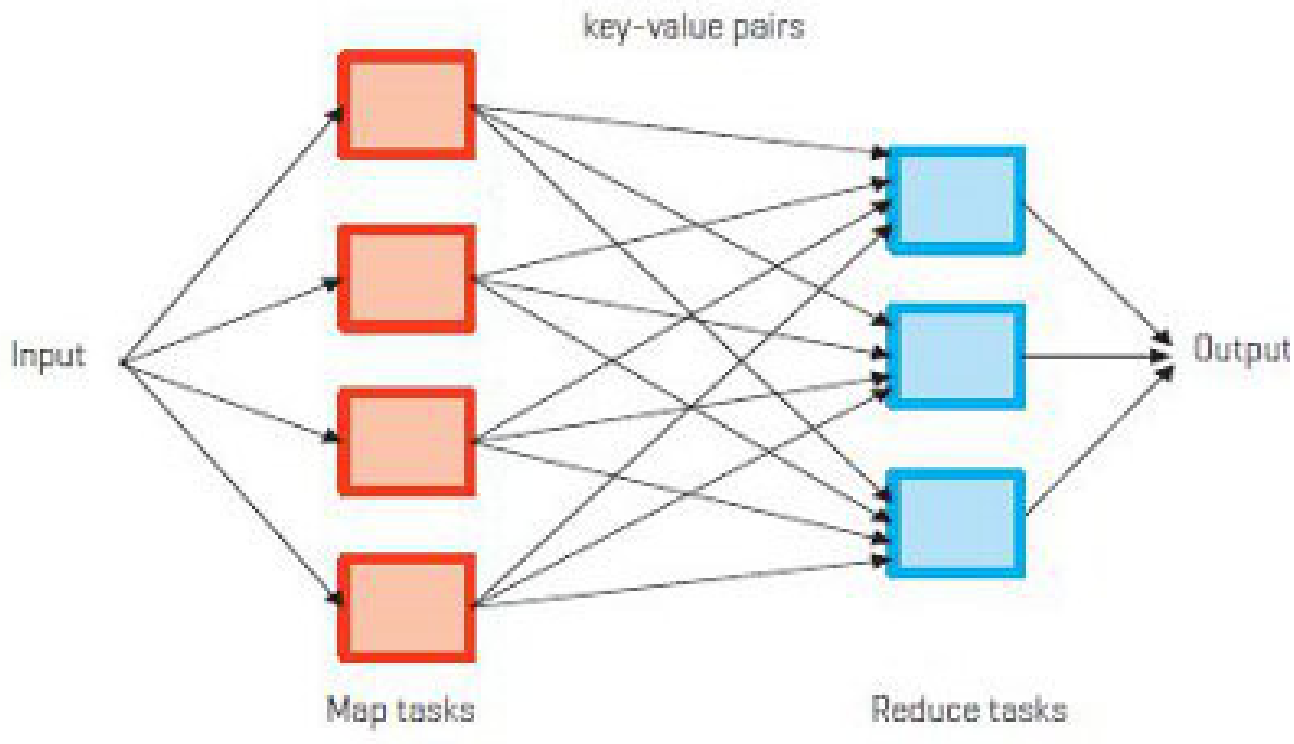
Логическая модель: Reduce

- Функция свертки (*reduce*) пережевывает свой вход и выплевывает ключ и некоторое итоговое значение
- $reduce_k : R_k \to (k, result_k)$
- Свертка может сделать что-то еще, например записать R_k на диск или в БД

Макет Map-Reduce

```
input_data = { 1: ["foo", "bar", "foo", "baz"] }
map_out = {}

def mapfn(docid, data, out):
    for word in data:
        if not word in out:
            out[word] = []
        out[word].append(1)

def reducefn(key, values):
    return key, sum(values)

for docid in input_data:
    mapfn(docid, input_data[docid], map_out)

for k in map_out.keys():
    k, v = reducefn(k, map_out[k])
```

Схема работы



Что в плюсе

- Простая модель программирования
- Хорошие задачи с шардируемыми входом и выходом легко масштабируются
- Код функций при этом не меняется

Что настораживает

- Не все задачи так хорошо шардируются
- Куча всевозможных форматов данных
- Как дирижировать работой сотен задач map и reduce?
- Кто же напишет shuffle?
- И где взять столько процессорных ядер?

Обязанности программной среды

- Реализовать shuffle
- Реализовать диспетчера, рассылающего задания мапперам, координирующего сортировку и свертку
- Написать чтение и запись разных форматов данных

Обязанности диспетчера

- Запуск рабочих процессов
- Распределение задач по процессам
  - равномерно распределять нагрузку
  - учитывать локальность данных
- Обработка сбоев
- Предоставление информации о ходе процесса

На первый взгляд все очень сложно

На первый взгляд все очень сложно

ничего, будет и второй взгляд

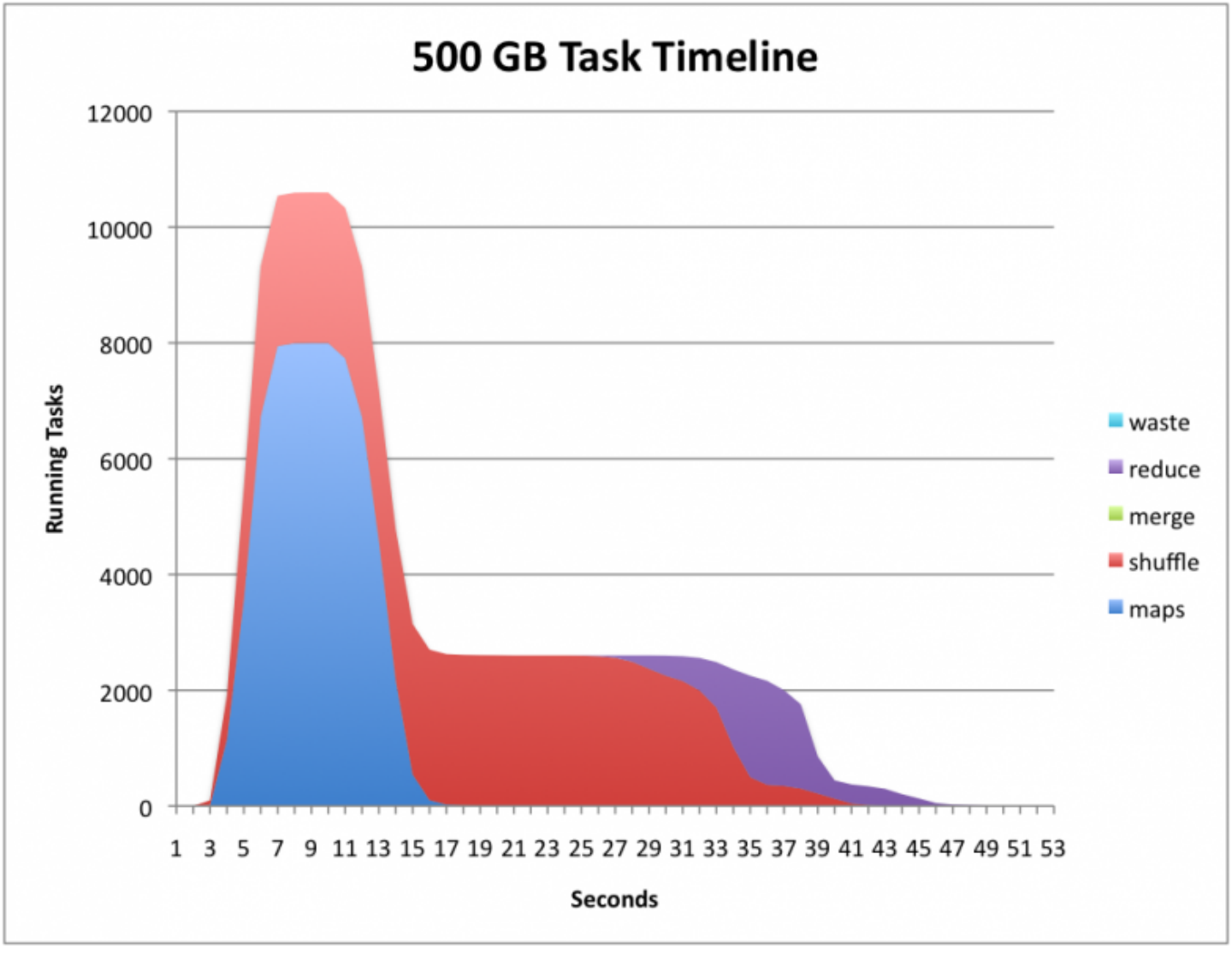
Между map и reduce

- Маппер разбивает свой выход на несколько частей в зависимости от функции *partition*
- *Partition* – хеш-функция: одинаковые ключи должны попасть в одну корзину
- Каждая корзина будет отправлена одной свертке
- Маппер записывает свои результаты на локальный диск и сообщает диспетчеру о каждой корзине
- Диспетчер звонит задачам свертки и сообщает где им брать данные

Между map и reduce

- Shuffle может идти параллельно с map
- Стадия reduce не может начаться, пока не закончится стадия map
- Если отказал маппер
  - текущую задачу надо переделать
  - прошедшие переделать если результаты не на GFS и reduce не успел их скопировать себе
- Если отказал reduce
  - текущую задачу надо переделать
  - результаты прошедших обычно лежат в GFS

Map-Reduce Art



Сегодня в программе

Map-Reduce с точки зрения программиста

Некоторые задачи

- Расширение модели

Map-Reduce на коленке

Подсчет и суммирование

- Документ – список или множество слов
- Нужно посчитать сумму вхождений каждого слова
- Решение: маппер выплевывает пары (слово, 1), свертка суммирует единички
- Напоминает `SELECT a, SUM(b) FROM T FROUP BY a`

Группировка

- Исходным документам сопоставлено некоторое множество характеристик
- Требуется для каждой характеристики найти и записать все документы, ею обладающей
- Решение: маппер выплевывает пары (характеристика, документ), свертка просто записывает списки в хранилище

Группировка: пример

- Характеристика – вхождение слова в текст документа
- Получаем для каждого слова список содержащих его документов
- Характеристика – домен первого уровня
- Получаем для каждого домена список страниц с сайтов в этом домене

Операции выборки и проекции

- Требуется выбрать записи по какому-то критерию и, возможно, каждую запись как-то преобразовать
- `SELECT a, b FROM T WHERE c=2`
- Маппер все делает и выплевывает преобразованные записи с каким-то ключом
  - если ключ свертки = ключу записи то в каждой задаче свертки будет 1 запись
  - если ключ свертки = имени отношения то будет 1 задача свертки на всё отношение
  - если ключ свертки = записи то мы удачно попаразитируем на среде выполнения и удалим дубликаты
- Другой вариант: преобразование делается в свёртке

Теоретико-множественные операции

- Объединение, пересечение, разность
- Маппер выплевывает пары (элемент множества, ID множества)
- Свертка:
  - в случае объединения – записывает ключ свертки 1 раз
  - в случае пересечения – записывает ключ свертки если в задаче свертки есть все исходные множества
  - в случае разности – записывает ключ свертки если в задаче свертки есть первое множество но нет второго

Соединение

- Требуется из двух множеств найти пары записей с одинаковым значением какого-то поля *a*
- Маппер выплевывает «тройки» (значение поля, (запись, ID множества))
- Свертка разделяет полученные записи по ID множества и строит декартово произведение

Комбайнер

- Если функция свертки коммутативная и ассоциативная, то свертку частично можно произвести в задаче маппинга
- Это делает комбайнер

Вторичный ключ свертки

- Для некоторых задач полезно отсортировать задачу свертки по какому-то критерию
  - тем более что shuffle все равно это делает
- В такой модели выходом маппера могут быть тройки (key, sec_key, value)
- Задача свертки формируется из троек с одинаковым ключом, но элементы отсортированы по значению sec_key

Сегодня в программе

Map-Reduce с точки зрения программиста

Некоторые задачи

- Расширение модели

Map-Reduce на коленке

Mincemeat: бомжатский Map-Reduce

- 450 строчек на питоне (диспетчер и сортировка)
- Код функций mapreduce – несколько строчек
- Любой компьютер в любой момент может стать частью кластера, достаточно знать IP диспетчера и пароль

Прикладная задача

- Раздобыли часть корпуса Google Books n-grams: 2-граммы начинающиеся с букв ”gr”
- 1G текста
- Хотим посчитать все начальные буквенные униграммы и биграммы в слове после слова ”great”

```
great britain

great wall

great work

"b": 1 "w":2 "br": 1, "wa": 1, "wo": 2
```

Забег первый

- На вход самому первому мапперу будет подаваться одна строчка из файла

```
Great persons\_NOUN      1768    1       1
```

- Маппер будет выплевывать единичку для униграммы и биграммы если строчка начинается с ”great”

Забег первый

- На вход самому первому мапперу будет подаваться одна строчка из файла

```
Great persons\_NOUN      1768    1       1
```

- Маппер будет выплевывать единичку для униграммы и биграммы если строчка начинается с ”great”

Поехали!

Забег первый: результаты

- Разминочный корпус состоит из 4M текста
- Один маппер справился с 4M за минуту с небольшим
- Как-то медленно, не?
- Эталонная программа безо всякого map-reduce справилась за полсекунды

Забег первый: результаты

- Разминочный корпус состоит из 4M текста
- Один маппер справился с 4M за минуту с небольшим
- Как-то медленно, не?
- Эталонная программа безо всякого map-reduce справилась за полсекунды

Эээээээ ...

Забег второй

- Давайте будем давать мапперу на вход все содержимое файла
- Маппер будет суммировать число найденных униграмм и биграмм и выплевывать на выход все полученные пары

Забег второй

- Давайте будем давать мапперу на вход все содержимое файла
- Маппер будет суммировать число найденных униграмм и биграмм и выплевывать на выход все полученные пары
- 1 секунда – уже лучше!
- Как будет на 1G?

Забег второй

- Давайте будем давать мапперу на вход все содержимое файла
- Маппер будет суммировать число найденных униграмм и биграмм и выплевывать на выход все полученные пары
- 1 секунда – уже лучше!
- Как будет на 1G?
  - Эталонная программа прошла за минуту с небольшим

Забег второй

- Давайте будем давать мапперу на вход все содержимое файла
- Маппер будет суммировать число найденных униграмм и биграмм и выплевывать на выход все полученные пары
- 1 секунда – уже лучше!
- Как будет на 1G?
  - Эталонная программа прошла за минуту с небольшим
  - Один mapreduce процесс справился за полторы минуты
  - Значит надежда есть

Забег третий: два процесса

- Добавим еще один mapreduce на соседней машине
- Погнали!

Забег третий: два процесса

- Добавим еще один mapreduce на соседней машине
- Погнали!
- Внезапно стало хуже...

Забег третий: два процесса

- Добавим еще один mapreduce на соседней машине
- Погнали!
- Внезапно стало хуже...
- Но два mapreduce процесса на одной двухядерной машине почти не проиграли эталону

Забег третий: два процесса

- Добавим еще один mapreduce на соседней машине
- Погнали!
- Внезапно стало хуже...
- Но два mapreduce процесса на одной двухядерной машине почти не проиграли эталону
- Когда процессов больше чем ядер, существенного прироста не наблюдается

Забег четвертый: равные условия

- Пересылать десятки мегабайтов по сети дольше чем читать с диска
- Пусть данные будут на каждой машине!
- Входом для маппера будет имя файла, а содержимое пусть читает сам

Забег четвертый: равные условия

- Пересылать десятки мегабайтов по сети дольше чем читать с диска
- Пусть данные будут на каждой машине!
- Входом для маппера будет имя файла, а содержимое пусть читает сам
- Результат:

Забег четвертый: равные условия

- Пересылать десятки мегабайтов по сети дольше чем читать с диска
- Пусть данные будут на каждой машине!
- Входом для маппера будет имя файла, а содержимое пусть читает сам
- Результат:
  - один mapreduce почти не отстал от эталона

Забег четвертый: равные условия

- Пересылать десятки мегабайтов по сети дольше чем читать с диска
- Пусть данные будут на каждой машине!
- Входом для маппера будет имя файла, а содержимое пусть читает сам
- Результат:
  - один mapreduce почти не отстал от эталона
  - вдвоем уже обогнали

Локальность данных

- Хорошо когда данные и код живут рядом. На одной машине
- Копировать все данные неразумно
- Возможные решения?

Локальность данных

- Хорошо когда данные и код живут рядом. На одной машине
- Копировать все данные неразумно
- Возможные решения?
- распределенные файловые системы

Эта презентация сверстана в

PAPEERIA

LaTeX в вашем браузере

alpha.papeeria.com

Литература I


Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: Simplified data processing on large clusters.
COMMUNICATIONS OF THE ACM, 51(1):107, 2008.