MAP REDUCE

Горских А.Г. ВМИ - 115
Рогов А.А. ВМИ - 115

Параллельное и распределённое программирование

- Под параллельным программированием понимают:
  - Векторную обработку данных
  - Использование нескольких CPU на компьютере
- Под распределённым программированием понимают использование многих CPU распределённых по разным компьютерам сети

Мотивация распределённых вычислений

- Хотим обрабатывать большие объёмы данных (> 1 ТВ)
- Хотим использовать мощности сотен/тысяч CPUs
- Хотим делать это быстро

Возникающие проблемы

- Отказы компьютеров
- Отказы сети
- Медленная коммуникация между компьютерами
- Пропускная способность канала ограничена
- Отсутствует глобальное состояние
- Компьютеры и сеть гетерогенны, не доверены и могут измениться в любое время

Идеи и решение

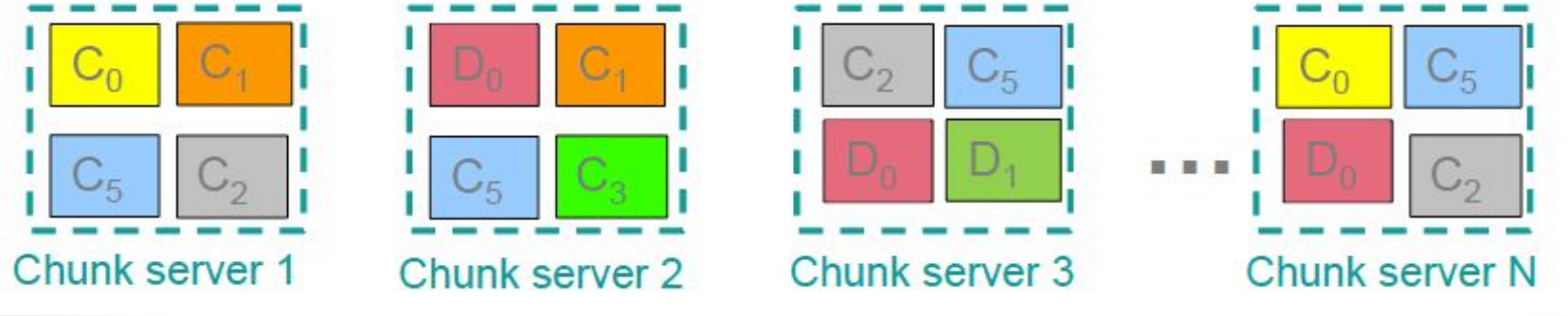
- Идеи
  - Перенести вычисления ближе к данным
  - Максимально снизить сетевые коммуникации
  - Средство контроля распределенных вычислений
  - Сохранить файлы несколько раз для надежности

- Решение от Google
  - 2003 год Google File System
  - 2004 год Map Reduce

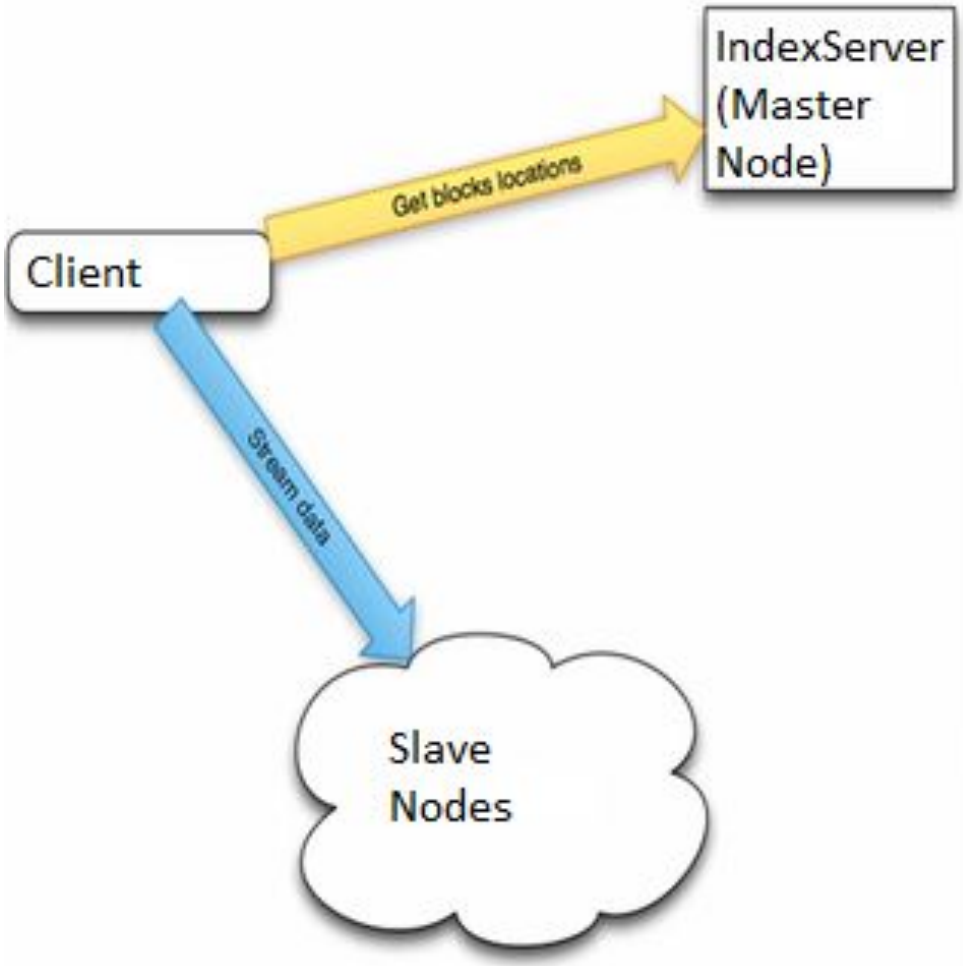
Распределенная файловая система

- Chunk Server (Slave Node)
  - Файл разделен на блоки (chunk)
  - Типичный размер блока 16-64 Mb
  - Каждый блок реплицируется на несколько машин

- Index Server (Master Node)
  - Хранение мета данных



Распределенная файловая система



Map Reduce

- Автоматическое распараллеливание и распределение по нодам
- Устойчивость к сбоям
- Автоматичексое управление внутренней коммуникацией между машинами
- Существование инструментов проверки и мониторинга
- Прозрачная абстракция для программистов

Идеология Map Reduce

- Идеология Map Reduce базируется на 2-х основных парадигмах:
  - Парадигме функционального программирования
  - Парадигме Master/Workers

Функциональное программирование

- Функции не изменяют данные – они всегда создают новые
- Оригинальные данные всегда существуют в нетронутом виде
- Порядок выполнения операций значения не имеет

Пример

```
fun foo(l: int list) =
  sum(l) + mul(l) + length(l)
```

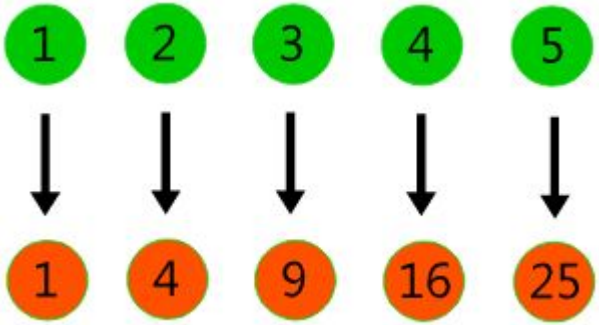
Порядок функций sum(), mul() и т.д. значения не имеет – Все они не изменяют значение переменной l

Map

- **Map f lst** – создает новый список, применив f к каждому элементу списка lst

Пример:

- Square x = x * x
- Map Square [1, 2, 3, 4, 5]

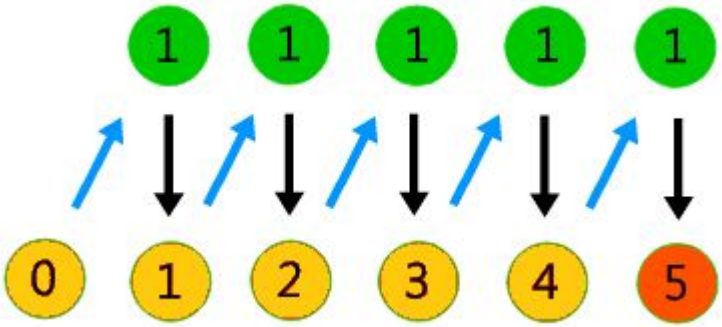


Reduce

- **Foldl f x0 lst** – свертка структуры данных к единственному значению
- x0 – аккумулирующее значение

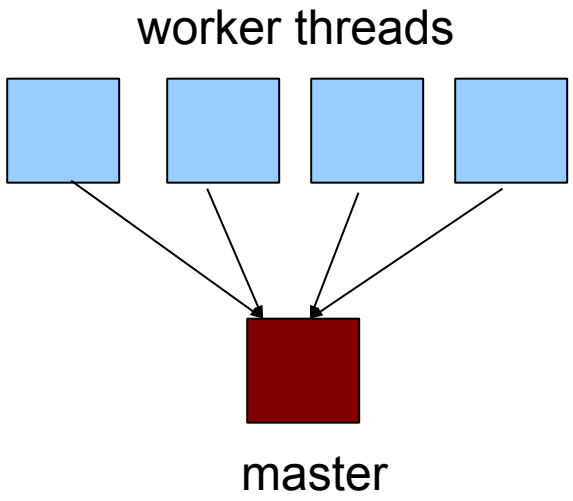
Пример:

- Sum(x, y) = x + y
- Foldl Sum 0 [1, 1, 1, 1, 1]



Master/Workers

- Есть один главный процесс, порождающий несколько рабочих процессов для обработки отдельных элементов данных.
- Управляет рабочими
- Ждёт возвращаемого рабочими результата
- Обеспечивает отказоустойчивость
- Реплицирует результаты свертки



Поток данных в MapReduce моделе

- Считывается большой набор данных
- **Map**: извлекаем необходимую информацию
- Shuffle and sort: на узле свертки ожидаются отсортированные ключи со списками значений
- **Reduce**: агрегация, фильтрация, трансформация
- Запись результатов

Модель программирования

- Заимствована из функционального программирования
- Пользователь реализует две функции:
  - **map (in_key, in_value) ->**
    (out_key, intermediate_value) list
  - **reduce (out_key, intermediate_value list) ->**
    out_value list

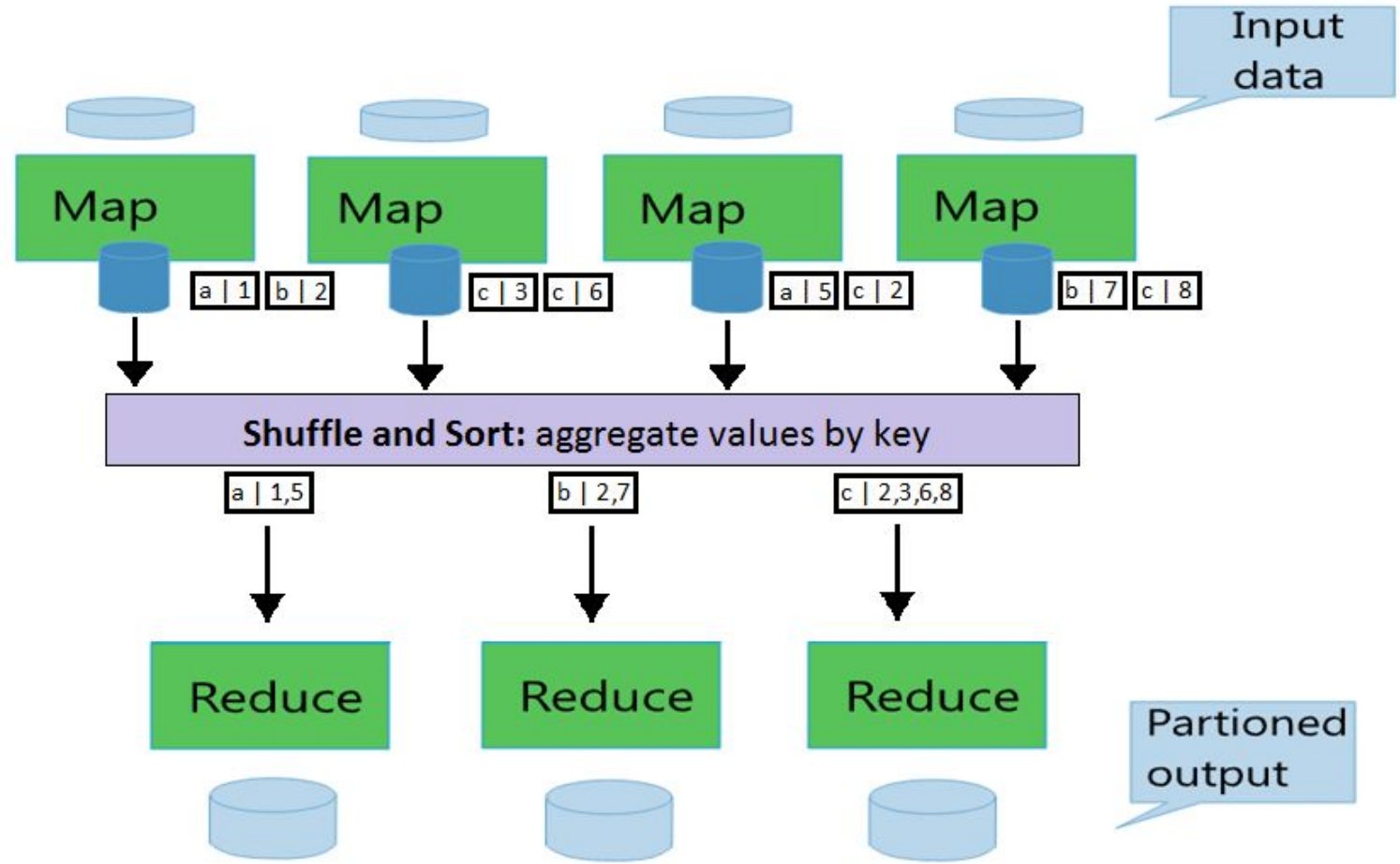
Функция map

- На вход функции поступают данные в виде пар ключ-значение. Например данные из текстового файла представляют собой. Кортежи вида (имя файла, строка файла).
- map() создаёт одно или несколько промежуточных значений, используя выходной ключ, переданный на вход.

Функция reduce

- После завершения стадии map’а все промежуточные значения для каждого выходного ключа добавляются в список
- reduce() комбинирует эти промежуточные значения в одно или более значений для каждого одинакового ключа
- На практике обычно по одному значению для каждого выходного ключа

MapReduce: workers



Параллелизм

- Функции map() выполняются параллельно, создавая различные промежуточные данные для различных входных групп данных
- Функции reduce() также выполняются параллельно, каждая работая над своим выходным ключом
- Все значения обрабатываются независимо
- Узкое место: фаза reduce не может быть начата, пока не завершится фаза map

Локальность

- Главная программа разбивает задачи основываясь на расположении данных: старается запускать map функцию на той же машине, где лежат данные.
- Входные данные для функции map разбиваются на блоки размером 64 MB (Это размер блока файловой системы Гугла)

Устойчивость к сбоям

- Главная программа обнаруживает отказы рабочих нодов и перезапускает задачи. Также происходит повторный запуск медленно выполняющихся заданий
- Главная программа запоминает конкретные пары ключ/значения, вызывавшие сбои и пропускает их при повторном запуске задач. Как результат – обходит ошибки в сторонних библиотеках!

Оптимизация

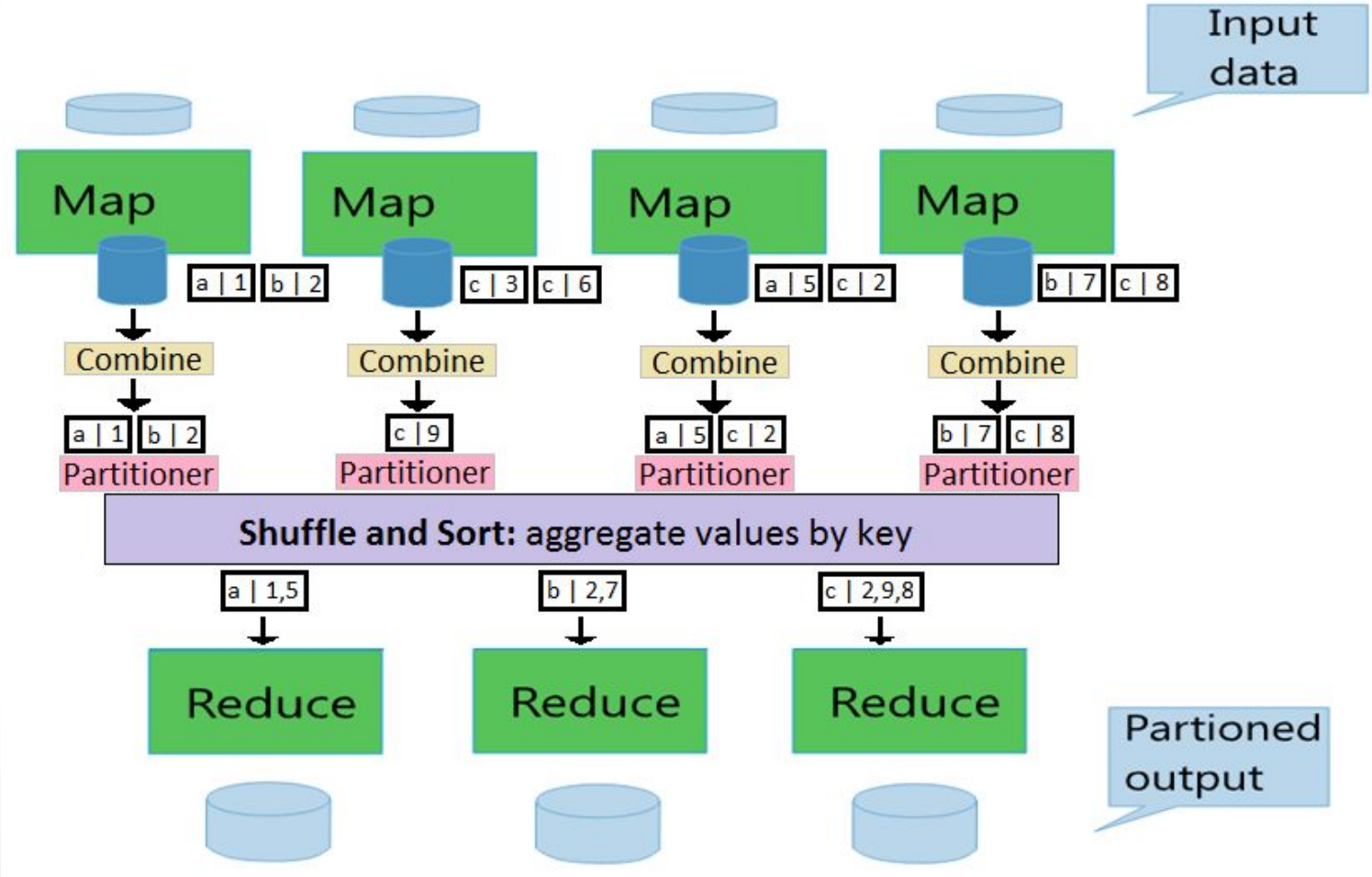
- Фаза reduce не может начаться пока не закончена фаза map. Один медленный диск может замедлить весь процесс.
- Поэтому главный процесс повторно выполняет медленно выполняющиеся задачи. Использует результаты первого завершившегося.

Оптимизация

Расширение набора пользовательских функций:

- Partition(ключ, кол-во reduce узлов) => reduce узел для данного ключа
  - Часто вычисляется как хэш ключа (Hash(k) mod n)
  - Разделяет пространство ключей для параллельного выполнения свертки
- Combine(ключ, список значений) => (ключ, значение)
  - Мини reduce, выполняется после map фазы на том же узле
  - Ипользуется для понижения трафика в сети

MapReduce: workers (opt.)



Пример: подсчет статистики по словам

```
Map(string input_key, input_value):
// input_key: document name
// input_value: document contents
For each word w in input_value:
    EmitIntermediate(w, “1”);

Reduce(string output_key, Iterator intermediate_values):
// output_key: a word
// intermediate_values: a list of counts
Int result = 0;
For each value v in intermediate_values:
    result += ParseInt(v);
Emit(AsString(result));
```

Пример: YAHOO web graph

- Для каждой странички формируетя список веб документов, ссылающихся на эту страничку
- На входе: веб документы
- Map: (doc_name, content) =>

  (href, {doc_name, link_text}) список
- Reduce: (href, [{doc_name1, link_text1}, …]) =>

  некоторая фильтрация (спам и т. д.)
- На выходе: таблица вида {target_url, source_url, link_text}

Пример: Last.fm top list

- На проигрыватель установлен плагин Last.fm
- Пользователь слушает песню => пишется лог вида

  {user, band, track}
- На входе: лог файлы
- Map: (log_name, log_data) => (user_band_tr, 1) список
- Reduce: (user_band_tr, [1, .. 1]) => сумма элементов списка
- На выходе: топ листы прослушиваемых треков для каждого пользователя

Реализации

- Google
  - Недоступна вне Google
  - GFS
- Hadoop
  - Открытая имплементация на Java
  - HDFS
- Aster Data
  - Cluster-optimized SQL Database которая также реализует MapReduce
- …

Решаемые задачи

- Индексация интернета
- Задачи исследования данных
- Data Mining данных
- Задачи построения отчетов
- Рендеринг набора кадров высококачественной анимации
- Симуляция нескольких сотен тысяч персонажей
- Симуляция интернета(PlanetLab)
- Ускорение скорости доставки контента(Akamai)

Будущее

- Microsoft Dryad – развитие идей map reduce.
- Программист определяет ацикличный направленный граф с C++ кодом в каждой вершине.
- Каждая работа может иметь множество входных и выходных потоков.
- Dryad занимается тем, что:
  - Определяет когда выполнять задачи
  - Где их выполнять
  - Восстанавливает компьютер после сбоя
  - Соединяет входы с выходами

Язык диаграмм Dryad

- G^n = параллельный запуск n копий G
- A >= B = подключить входы B к выходам A
- A>>B = подключить каждую работу в A к работе в B
- A || B = объединение работ

  - Например, а диаграмма MapReduce может записана на языке Dryad как Mapper^n >> Reducer^m .
  - Dryad также позволяет указывать как реализовать каждой ребро: как файл, TCP pipe или FIFO на общей памяти.

Заключение

- MapReduce доказал свою эффективность
- Сильно упростил распределённые вычисления в компании Google
- Парадигма функционального программирования может применяться к распределённым вычислениям.
- Лёгкость использования – позволяет сосредоточиться на проблеме, а не на деталях реализации