

Пределив, где и как можно хранить большие данные, нужно предложить способ для их эффективной обработки. Следует отдельно отметить, что HDFS налагивает значительные ограничения на операции над данными (например, запрет произвольного доступа), что, в свою очередь, оказывает влияние на те алгоритмы, которые будут работать эффективно поверх такой файловой системы (например, мы не можем индексировать данные произвольным образом).

Основным подходом к обработке больших данных, хранимых в HDFS можно считать фреймворк MapReduce.

I. Мотивация.

1. Значительно параллельные задачи.

Оказывается, что многие задачи связанные с обработкой больших объемов данных могут быть

разбиты на полностью независимые, параллельные подзадачи.

Например:

a) По корпусу документов

$D = \{d_1, \dots, d_n\}$ построить

индекс для слов из этих документов. $\{w_i \rightarrow [d_{i1}, \dots, d_{in}]\}$.

Каждый документ может быть обработан независимо, а затем результаты объединены.

b) Предобработка данных, например, подсчитать слов в каждом документе из D — все документы независимы.

2. Использование простых, но мощных примитивов.

В функциональном программировании для работы со списками зачастую используются функции map и reduce со следующей семантикой:

- $\text{map}(f, [v_1, \dots, v_n]) \rightarrow [f(v_1), \dots, f(v_n)]$

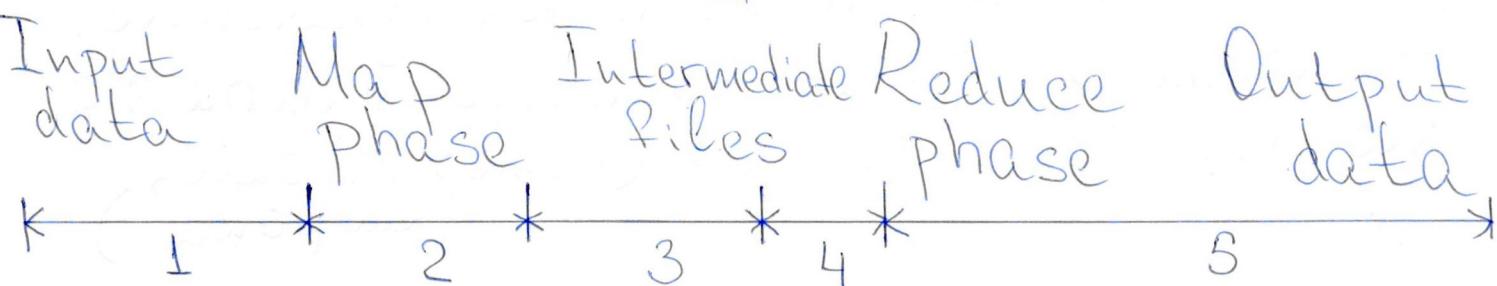
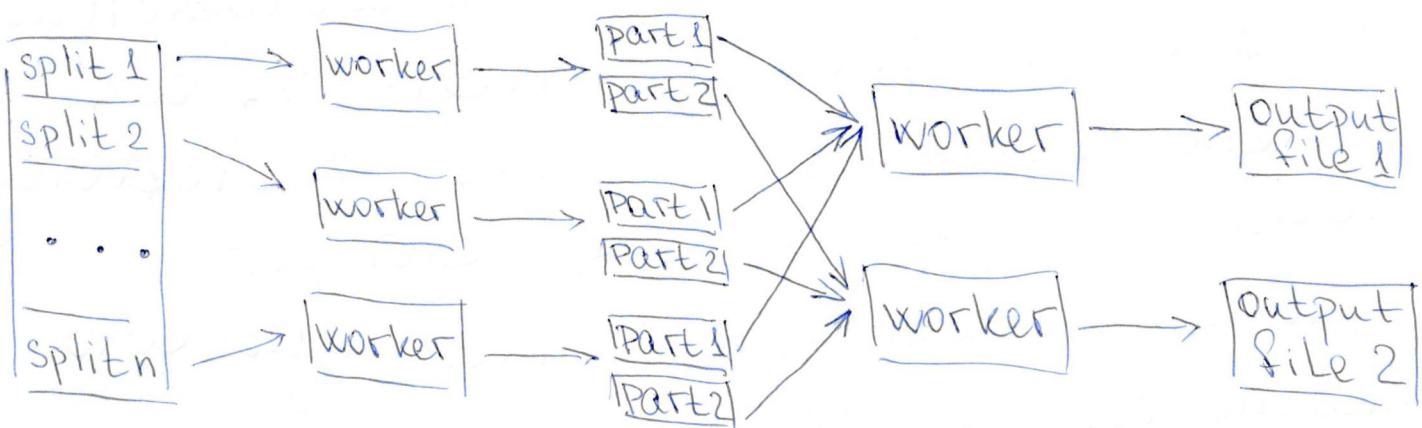
Т.е. преобразование списка
в новый список.

- $\text{reduce}(g, [v_1, \dots, v_n]) \rightarrow g(v_1, \text{reduce}(g, [v_2, \dots, v_n]))$

Т.е. преобразование списка
в некий один объект.

Map Reduce разбивает эти примитивы.

II Map Reduce Flow.



Разберём кампаний этап:

1. Всё начинается с входных файлов.
Это может быть как один файл, так и их набор.

Основные требования к входным данным:

- Находятся в HDFS.
- Допускают разбиение на независимые блоки (сplits).

Так, например, это может быть текстовый файл, это может быть архив, допускающий разбиение (bzip2, но не gz, tar).

Файлы, поданное на вход разбиваются на splits, обычно в соответствии с размером блока в HDFS, и, затем подаются на вход соответствующим воркерам фазы 2.

При этом, каждый split суть есть множество из пар ключ-значение. Например, для текстового файла это может быть пара (номер строки, строка)

2. На вход мапперу, последовательно подаются пары ($K=V$) из соответствующего списка с фазы 1.

Маппер преобразует полученные данные и возвращает такие списки

нап (K-V). Этот список не обзан быть
той же длины, тока, что и входной.

Важно отметить, что все маниперы
работают полностью независимо и
не обмениваются информацией.

3. Вход манипера разбивается на
части, каждая из которых соответст-
вует одному и только одному редью-
серу. Но упрощенно, разбиение
происходит по ключу: $\text{hash}(k) \% R$,
но может и зависеть от V.
Каждый из получившихся файлов
сортируется по ключу.

4. По результатам фазы 3, каждый
манипер сгенерирован по файлу на
каждом редьюсер. Теперь, для
каждого редьюсера, соответствующие
ему файлы объединяются, сохра-
няясь отсортированность, т.е. происхо-
дит слияние отсортированных
списков (merge).

И, наконец, получившийся отсортиро-
ванный список разбивается на

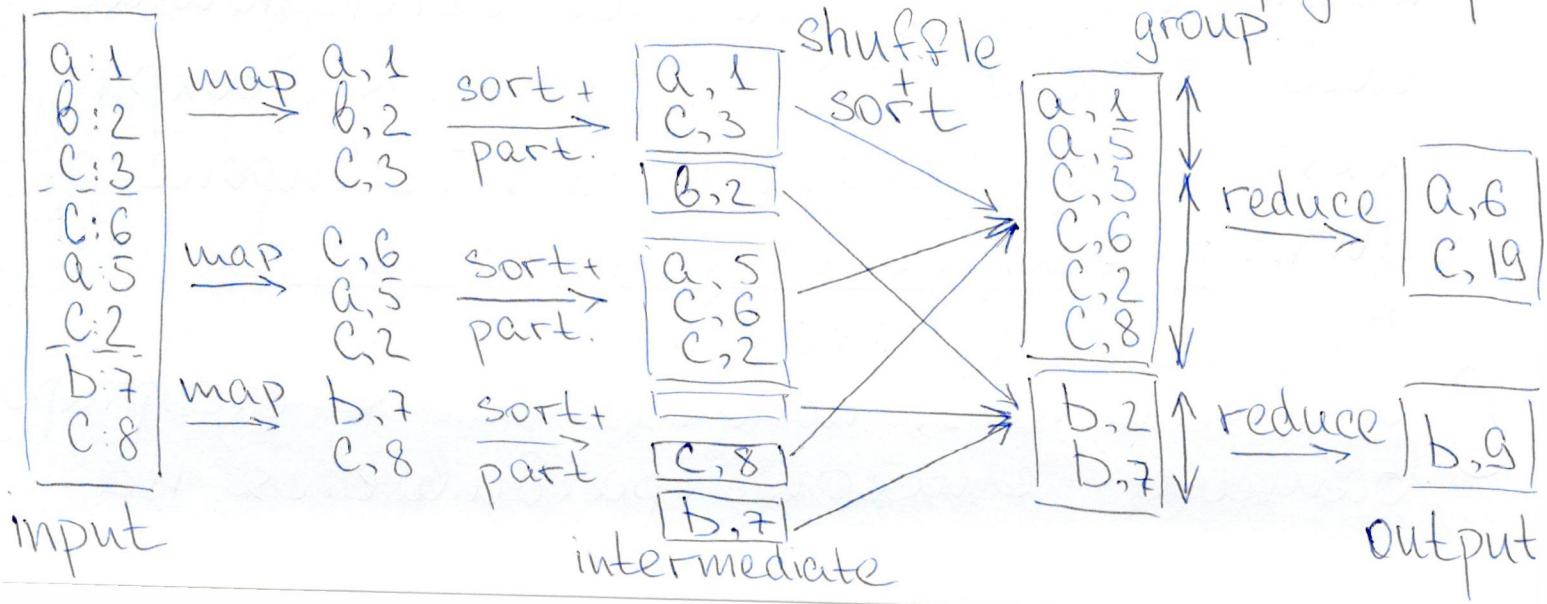
6

Коды для пар (K-V) с одинаковыми ключами, т.е. происходит группировка значений с одинаковыми ключами.

3. После группировки на фазе 4 пары ($K, [V_1, \dots, V_t]$) поочередно подаются на вход редьюсеру. Он, в свою очередь, преобразует и выдаёт glue кампой такой пары сразу же заменяя её в выходной файл для данного редьюсера.

Рассмотрим пример:

1. Файл содержит в себе множество строк формата "tag: num".
Нужно определить сумму чисел glue каждого типа. Пополним, размер блока HDFS = 3 строки; # редьюсеров = 2:



4. Если количество различных слов невелико и мы можем позволить Hadoop достаточно большими, то можно еще уменьшить объем данных на выходе маппера:

```
H = defaultdict(int) ] initialize
map(doc_id, doc): ]
    for word in doc: ]
        H[word] += 1
    for word, c in H.items(): ]
        Emit(word, c) ] close
```

Мы можем следить, чтобы размер H оставался недостаточным. Собирая данные из маппера при превышении порога по памяти. Однако, тогда нужно самостоятельно контролировать использование памяти.

VI Hadoop Streaming.

Hadoop предоставляет возможность писать MR задачи на Java, C++, scala, но не на python. Если же хотите использовать python для MR задач, то необходимо использовать обертки —

Hadoop Streaming.

Особенности:

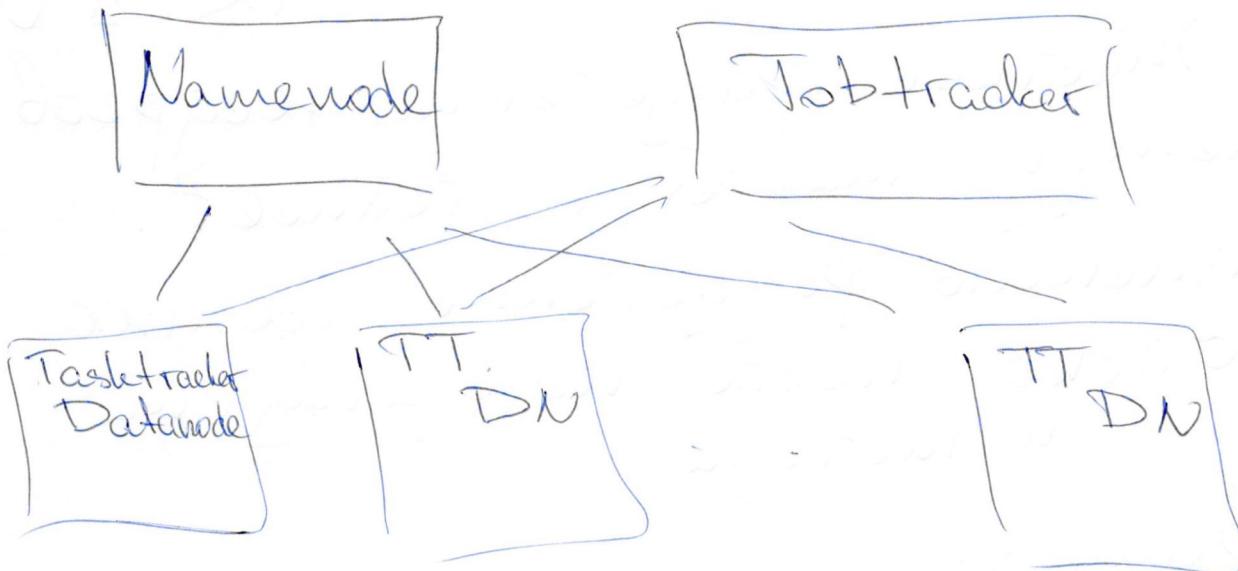
- Маппер и редьюсер получают данные из `stdin`, а "отдают" в `stdout`.
- Маппер обрабатывает вход построчно и выдает $K-V$ пару через табуляцию.
- Редьюсер принимает данные, отсортированные по ключу и выдает $K-V$ в стандартный выход.
- Работа Hadoop Streaming без дополнительных параметров в точности эквивалентна:
`cat input | map.py | sort | reduce.py`.
- С помощью соответствующих ключей можно задавать все 3 описанных ранее части MR программ.

Map Reduce (1.0)

Рассмотрим как организован запуск MR задач на кластере

HeadNode управляет с помощью бирюзовых блоков:

- Управление запуском задач
- Перемещение данных
- Синхронизация этапов MR.
- Обработка ошибок и отказов



1. Job Tracker

- Управляет запуском всех задач
- Мониторит прогресс выполнения

Задача обеспечивает восстановление утерянных задач.

- Используется узким местом

2. Task Tracker

- Запускает боркеры на свой машине
- Общение с JT, передача статистики, состояния боркеров

Основной недостаток MR 1.0.

— местное разпределение ресурсов между mapper и reducer.

Изменить разпределение можно только если перезапустить мастер.

Пример:

- 10 серверов но 16 задач
- Если 160 задач, то 10 задачи под TaskTracker

- Задача требует 300 mappers и известно, что map работает за 10 минут, а результат обрабатывается на 1 reducer за 100 минут.

1. Максимальное параллелизм:

100 Map + 50 Reduce

$$\Rightarrow T = \frac{300}{100} \cdot 10 + \frac{100}{50} \approx 32.$$

2. Неравенство для задачи

$$\begin{cases} x + y = 150 \\ \frac{300}{x} \cdot 10 + \frac{100}{y} \rightarrow \min_{x,y} \end{cases}$$

$$\Rightarrow T^* \approx 28 \text{ мин.}$$

3. Осуществление параллелизма

$$10 \cdot \frac{300}{150} + \frac{100}{150} = 20 \text{ мин.}$$

Для решения проблемы параллелизма ресурсы для параллелизма YARN

