Особенности работы с большими объёмами данных

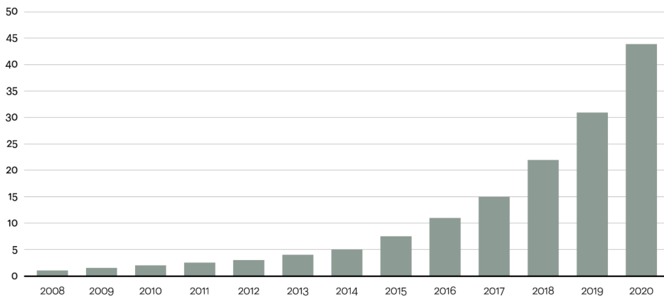
Остапец Андрей

1 октября 2014 г.

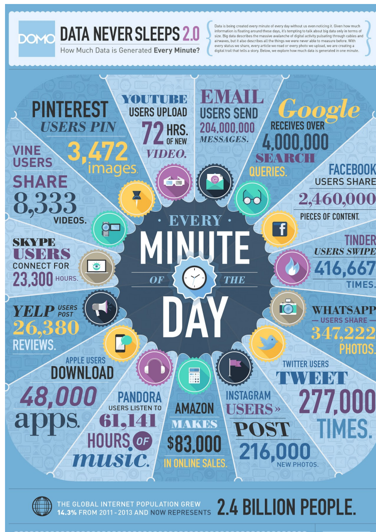
Сегодня поговорим о...

- Big Data
- Модель Map-Reduce
- Apache Hadoop
- Разработка Job(Java, Streaming, Hive)

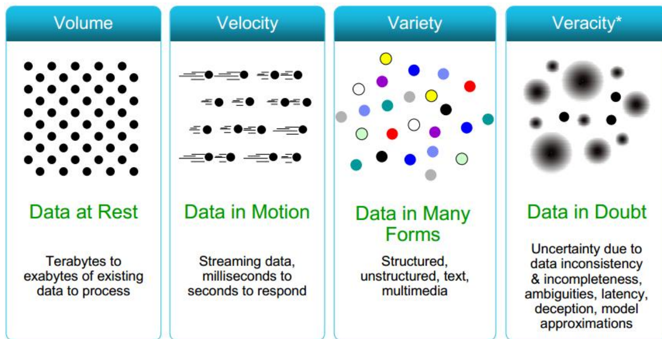
Data Explosion



Data Explosion



Three Vs



Что такое BigData?

- К BigData можно отнести такие коллекции данных размер которых лежит за гранью того, что могут хранить, обрабатывать и анализировать типичные СУБД
- Это определение может варьироваться в зависимости от того какое используется ПО и какие размеры данных приняты в данной среде
- С развитием технологий размер данных, которые можно определить как BigData, также меняется
- В зависимости от этого размер BigData может варьироваться от десятка терайбайт до десятков и сотен петабайт

MapReduce

Технология распределенная обработки большого объема данных

- Достаточно проста.
- Легко масшабируема.
- Применима для решения широкого круга задач.

История

- В 2003 году компания Google выпустила статью про distributed file systems, как они хранят у себя внутри данные и индексы, данные о пользователях и прочее.
- В 2004 году компания Google выпустила статью, которая описывает парадигму обработки такого объема данных, которая называется MapReduce.

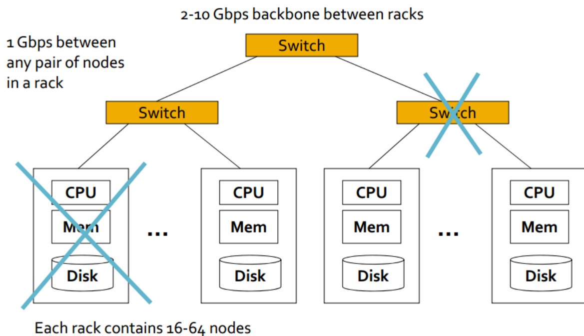
MapReduce Design Goals

1. Масштабируемость для больших объемов данных:
   - Тысячи машин, десятки тысяч дисков
2. Cost-efficiency:
   - Commodity clusters (обычные сервера, дешего, но ненадежно)
   - Обычая локальная сеть
   - Fault-tolerance «из коробки» (меньше надо сисадминов)
   - Простота использования (меньше программистов).

Commodity Clusters

1. Стандартная архитектура включает:
   - Кластер из обычных Linux машин
   - Сеть - Gigabit ethernet interconnect
2. Как организовать вычисления на такой архитектуре?
   - Надо скрыть проблемы «железа» от пользователей

Cluster architecture



Distributed File System

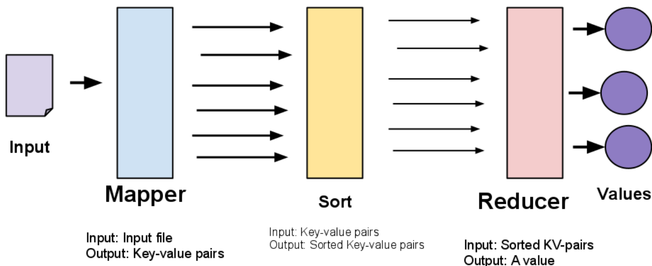
Задачи:

- Хранение больших объемов данных.
- Прозрачность.
- Масштабируемость.
- Надежность

Distributed File System

- Chunks:
  - Файл последовательно разбивается на чанки, типичный размер 64-512MB.
  - Каждый чанк реплицируется (обычно 3x).
  - Стараемся держать реплики на разных стойках.
- Master node
  - Обслуживает мета-данные файлов.
  - Репликация.
- Чтение клиента идут напрямую из slave nodes.

Map-Reduce Framework



Пример приложения: WordCount

- Большие файлы с документами (тексты)
- Подсчитать число вхождений каждого слова в файле
- Файл на диске - слишком много различных слов для того, чтобы все уместить в памяти.

WordCount на Map-Reduce

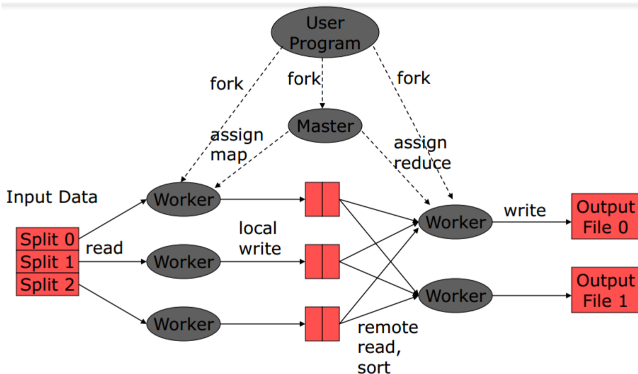
```
map(key, value):
// key: doc name; value: text of doc
  for each word w in value: emit(w, 1);

reduce(key, values):
// key: a word; values: list of counts
  result = 0;
  for (each count v in values)
     result += v;
  emit(key, result);
```

WordCount на Map-Reduce

- Программа выполняет fork master процесса и множества worker процессов.
- Ввод разбивается на некоторое число splits.
- Worker процесс назначается для выполнения либо функции Map используя split или функции Reduce для некоторого набора промежуточных данных.

Distributed Execution Overview



За что отвечает Master

1. Назначает таски Map и Reduce воркерам
2. Проверяет, что никто из воркеров не умер
3. Распределяет результаты фазы Map на Reduce

Взаимодействие между Map и Reduce

1. Выбрать число R reduce тасков.
2. Распределить промежуточные ключи в R групп (например, по хешу).
3. Каждый таск Map создает на своей стороне R файлов промежуточных пар key-value, по одному для каждого таска Reduce.

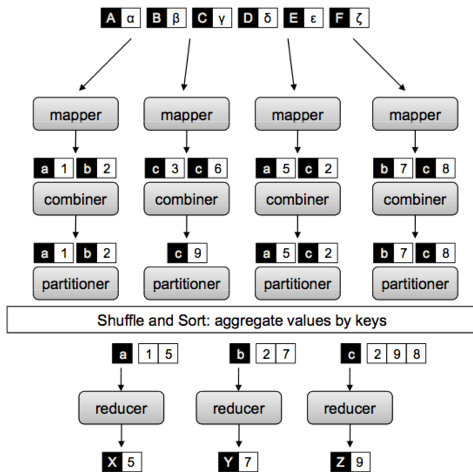
Failures

- Если процесс воркера падает, то все его таски переназначаются на другие воркеры.
- Master failure:
    - Задача Map-Reduce завершается с ошибкой и отправляется уведомление клиенту

Combiners

- Часто один таск Map генерит много пар $(k, v_1), (k, v_2), \ldots$ для одного и того же k
  - Пример: часто встречаемые слова в задаче Word-Count
- Используем пре-агрегацию в Map
  - Используем функцию Reduce на части данных от Map таска
- Работает только в случае, если функция Reduce коммутативная и ассоциативная

Map-Reduce Flow



Hadoop

- Open-source реализация Map-Reduce на Java.
- http://hadoop.apache.org/

Hadoop, зачем?

- Надо обрабатывать MultiPetabyte Datasets
- Сложно и дорого реализовать это в своем приложении.
- Failed Nodes происходит каждый день
  - Failure ожидаемы, это не должно быть неожиданностью.
  - Число нод в кластере непостоянно.
- Нужна общая инфраструктура
  - Эффективная, надежная, Open Source Apache License

Hadoop, история

- 2004 Дуг Каттинг и Майк Кафарелла создают первые реализации того, что позднее станет HDFS и Mapr-Reduce.
- 2005 – Nutch использует MapReduce. Hadoop надежно работает на 20 узлах.
- Февраль 2006 – Nutch становится подпроектом Lucene.
- Май 2006 - Yahoo! развертывает исследовательский кластер Hadoop из 300 узлов.
- Апрель 2007 – Yahoo! на 1000-node кластере.
- Jul 2008 – 4000 node test cluster

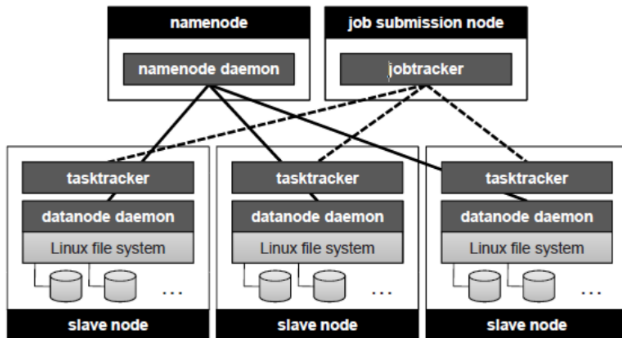
Hadoop, кто использует?

- Amazon
- Facebook
- Google
- IBM
- Last.fm
- Yahoo!
- Yandex
- Mail.ru

Глоссарий

- Job - «готовая программа», включающая в себя Mapper и Reducer
- Task - «часть программы», включающая либо Map или Reduce процесс

Hadoop architecture



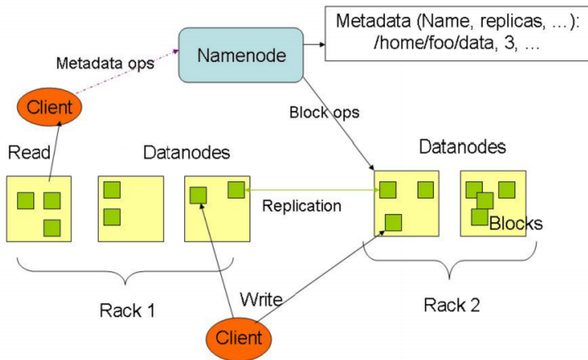
Задачи HDFS

- Очень большая Distributed File System
  - 10K nodes, 100 million files, 10 PB
- Подразумевает Commodity Hardware
  - Файлы реплицируются для того, чтобы справляться с hardware failure
  - Определять failures и уметь восстанавливаться после них
- Оптимизация для Batch Processing
  - Расположение данных такое, что код перемещается к данным, а не наоборот
  - Предоставляет высокую пропускную способность.
- User Space, запускается на различных OS

HDFS

- Single Namespace для всего кластера
- Data Coherency
  - Write-once-read-many access model
  - Клиент может только дописывать к существующим файлам
- Файлы разбиваются на блоки
  - Обычно 128 MB block size
  - Блок реплицируется на несколько DataNode
- Intelligent Client
  - Клиент может узнать расположение блоков
  - Клиент имеет доступ к данным прямо с DataNode

HDFS architecture



NameNode, Meta-данные

- Все meta-данные в памяти
- Типы meta-данных
  - Список файлов
  - Список блоков для каждого файла
  - Список DataNode для каждого блока
  - Файловые атрибуты, например время, фактор репликации
- Transaction Log
  - Записи о создании или удалении файла

DataNode

- Block Server
    - Хранит данные в локальной файловой системе (напр. ext3)
    - Хранит meta-данные блока (напр. CRC)
    - Отдает данные и meta-данные клиентам
- Block Report
    - Периодически отправляет отчет о своих блока на NameNode
- Pipelining of Data
    - Перенаправляет данные на другие DataNode'ы

Размещение блоков

- Классическая стратегия
  - Одна реплика на локальной ноде
  - Вторая реплика на удаленной стойке (rack)
  - Третья реплика на той же стойке
  - Дополнительные реплики размещаются произвольно
- Клиент читает с ближайшей реплики

Корректность данных

- Использование checksums для проверки данных
    - Используется CRC32
- Создание файла
    - Клиент считывает checksums на каждые 512 байт
    - DataNode хранит эти checksums
- Доступ к файлу
    - Клиент запрашивает данные и checksums от DataNode
    - Если проверка не пошла, то клиент делает запрос на другую реплику

NameNode Failure

- До последней версии - Single point of failure (Единая точка отказа)
- Transaction Log хранится на каждой ноде
- Дополнительная NameNode, которая хранит копии метаданных.

NameNode Failure

- Клиент запрашивает список DataNodes на которых будут находится реплики блока
- Клиент записывает блок на первую DataNode
- Первая DataNode переправляет данные на следующую DataNode в pipeline и т.д. (в зависимости от кол-ва реплик)
- Когда все реплики записаны клиент переходит к записи следующего блока файла

Fault Tolerance в MapReduce

Если таск падает:

- Перезапуск на другой ноде
  - OK для map потому что нет зависимостей и input в HDFS
  - OK для reduce потому что output от map на локальном диске
- Если один и тот же таск постоянно падает, то job failed или игнорируем этот блок входных данных (определяется пользователем)

Fault Tolerance в MapReduce

Если падает нода:

- Перезапустить запущенные на ней таски на других нодах
- Перезапустить все map-таски, которые отработали на этой ноде
  - Это обязательно потому что их output файлы были потеряны после падения самой ноды

Fault Tolerance в MapReduce

Если таск долго выполняется (straggler):

- Запустить копию таска на другой ноде (“speculative execution”)
- Использовать output того таска, который быстрее выполнится, и убить медленный

- Медленные или зависшие таски появляются довольно часто из-за проблем с железом, ошибок в коде, проблем конфигурации и т.д.
- Один единственный медленный таск может замедлить всю задачу

Итоги

- Достоинства:
  - Гладкая масштабируемость(для 2x производительности достаточно 2x оборудования (почти))
  - OpenSource
  - Удобен для research-задач
- Недостатки:
  - Высокая стоимость поддержки и администрирования
  - Необходим штат квалифицированных developer'ов
  - Нестабильность
  - Низкая скорость,
  - Не real-time

Итоги

- Java код. Приложение WordCount
- Hadoop streaming. Приложение WordCount
- Hive Query Language. Примеры запросов

Java код

```
public static class Map extends MapReduceBase implements
    Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new
        IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
        OutputCollector<Text, IntWritable> output, Reporter
        reporter) throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new
            StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
```

Java код

```
public static class Reduce extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable>
        values, OutputCollector<Text, IntWritable> output,
        Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

Java код

```
public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
}
```

Hadoop Streaming

- Технология разработки MR Job не на Java.
- На каждом сервере кластера устанавливается интерпретатор языка(Python, Perl, Ruby, ...).
- Отдельно пишем Map, отдельно Reduce.

Hadoop Streaming(Mapper)

```
import sys

for line in sys.stdin:
    line = line.strip()
    unpacked = line.split(",")
    stadium, capacity, expanded, location, surface, turf,
        team, opened, weather, roof, elevation = line.split(
        ",")
    results = [turf, "1"]
    print("\t".join(results))
```

Hadoop Streaming(Shuffle)

```
# Example input (ordered by key)
# FALSE 1
# FALSE 1
# TRUE 1
# TRUE 1
# UNKNOWN 1
# UNKNOWN 1

# keys come grouped together
# so we need to keep track of state a little bit
# thus when the key changes (turf), we need to reset
# our counter, and write out the count we've accumulated
```

Hadoop Streaming(Reducer)

```
last_turf = None
turf_count = 0
for line in sys.stdin:
    line = line.strip()
    turf, count = line.split("\t")
    count = int(count)
    # if this is the first iteration
    if not last_turf:
        last_turf = turf
    # if they're the same, log it
    if turf == last_turf:
        turf_count += count
    else: # state change (previous line was k=x, this line
        is k=y)
        result = [last_turf, turf_count]
        print("\t".join(str(v) for v in result))
        last_turf = turf
        turf_count = 1
print("\t".join(str(v) for v in [last_turf, turf_count]))
```

Hadoop Streaming

- Плюсы:
  - Простота разработки.
  - Удобство отладки.
  - Любой интерпретируемый язык.
- Минусы:
  - Не все возможности Hadoop.
  - Замедление примерно 15 %.
  - Сложности с бинарными данными.

Hive

- Фреймворк на базе Apache Hadoop
- Транслирует SQL запросы в MapReduce jobs
- Используется как основной R&D инструмент в Facebook
- http://demo.gethue.com/