Анализ больших данных с Apache Spark

Лекция 1. Парадигма MapReduce

Мурашкин Вячеслав
2017

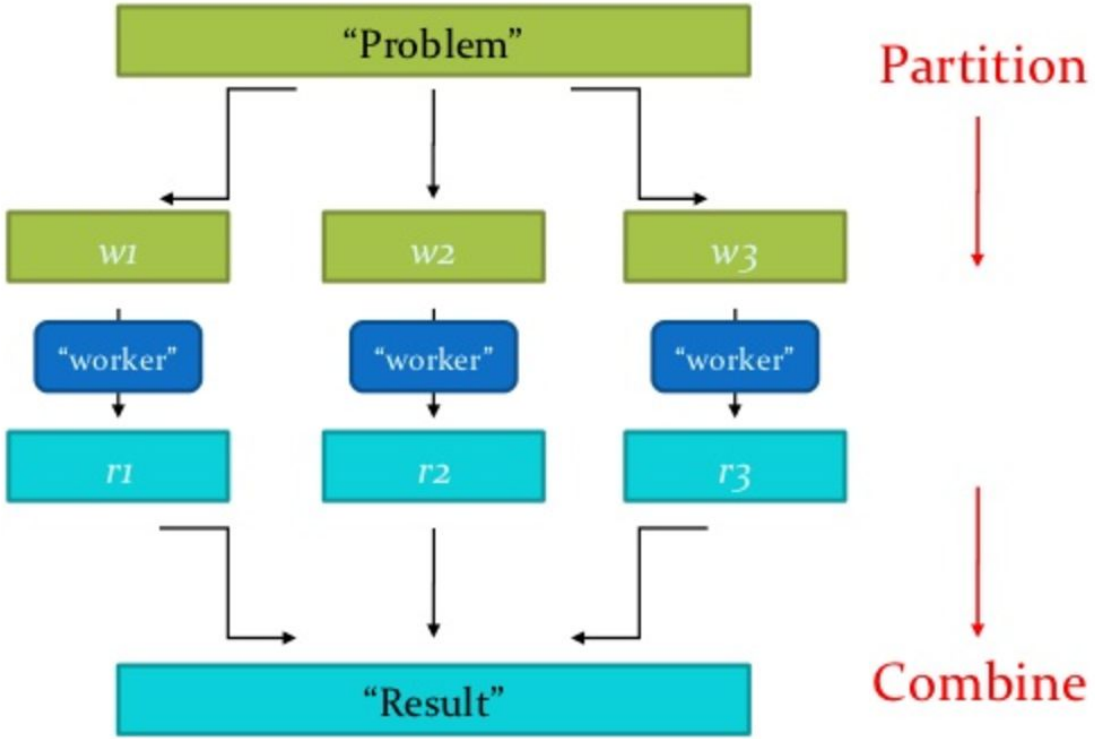
Лекция 1. Парадигма MapReduce

- Почему MapReduce?
- Парадигма MapReduce
- Распределенное хранилище: HDFS
- Экосистема приложений Hadoop
- Пример: подсчет числа слов в корпусе
- Пример: операции JOIN на MapReduce
- Пример: логистическая регрессия

Почему MapReduce?

- Алгоритмы работающие на одном сервере сложно масштабировать при увеличении объема данных
- В случае разделения вычислительных ресурсов и хранилища возникает узкое место при переносе данных, решение: перенести вычисления к данным
- Сервер с мощным CPU, как правило, стоит дороже, чем эквивалентное число более слабых машин
- Возникла потребность в **отказоустойчивом** решении для обработки большого объема данных **дешево** с простым способ **масштабирования** без изменения логики работы алгоритмов

Парадигма MapReduce



Парадигма MapReduce

- Данные на входе и выходе алгоритма всегда передаются в виде **набора** пар **<key, value>**
- Для обработки данных необходимо реализовать **две** функции:

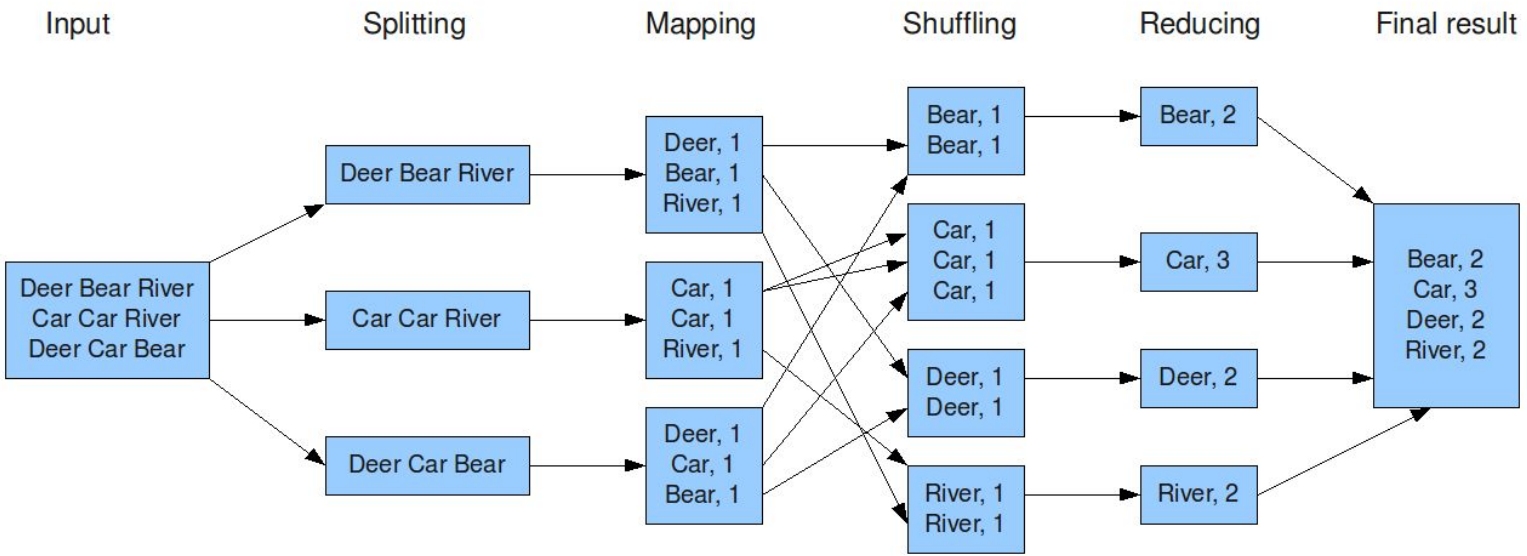
  map (in_key, in_value) -> list(out_key, intermediate_value)

  - принимает на входе пару <in_key, in_value>
  - возвращает список пар с промежуточными значениями <out_key, intermediate_value>

  reduce (out_key, list(intermediate_value)) -> list(out_key, out_value)

  - обрабатывает промежуточные значения для ключа out_key
  - возвращает результат обработки в виде списка (как правило из одного элемента)

Пример: подсчет частот слов в корпусе



Пример: подсчет частот слов в корпусе

```
map(String input_key, String 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; output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
  Emit(output_key, AsString(result));
```

Плюсы и минусы

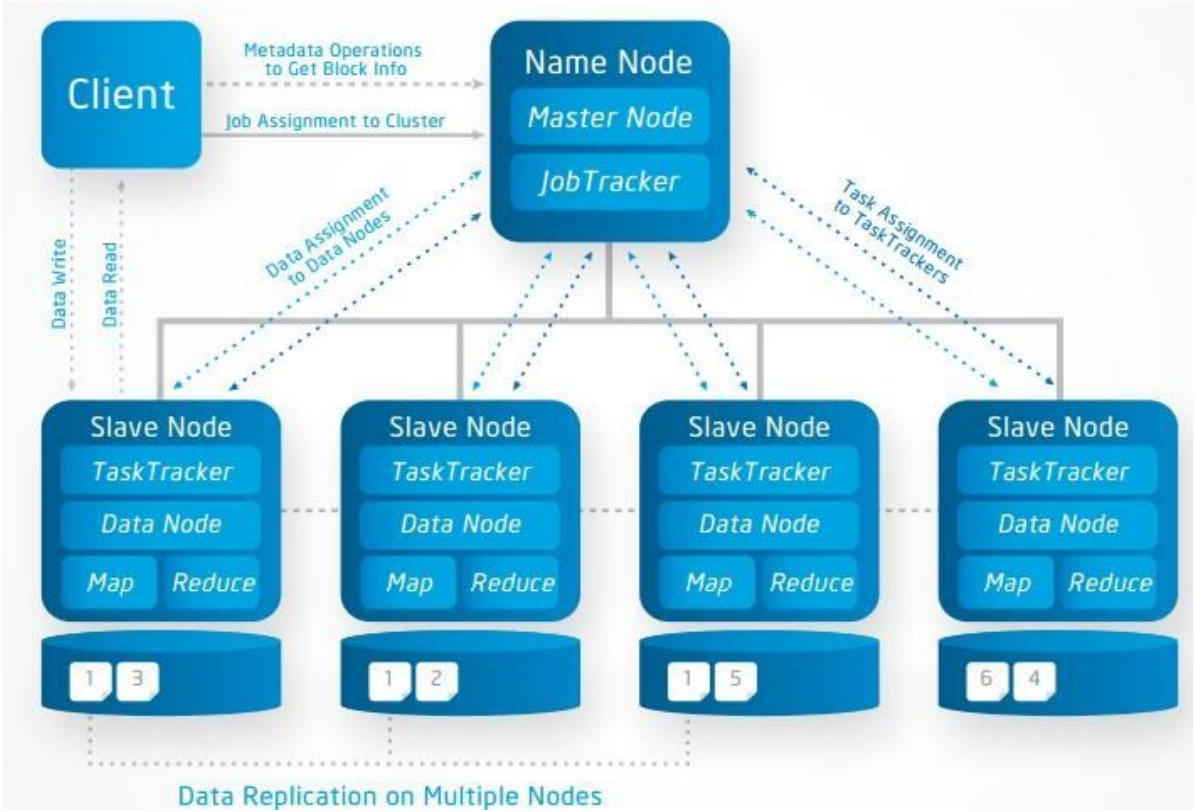
+ Map и reduce стадии легко параллелятся
+ Линейное масштабирование
+ Отказоустойчивость: независимая обработка данных, в случае падения одной машины не нужно перезапускать весь процесс заново

- Данные после каждого этапа сохраняются на диск (тратим время на запись/чтение данных)
- Необходимо копировать данные после map стадии (группировка по ключу)

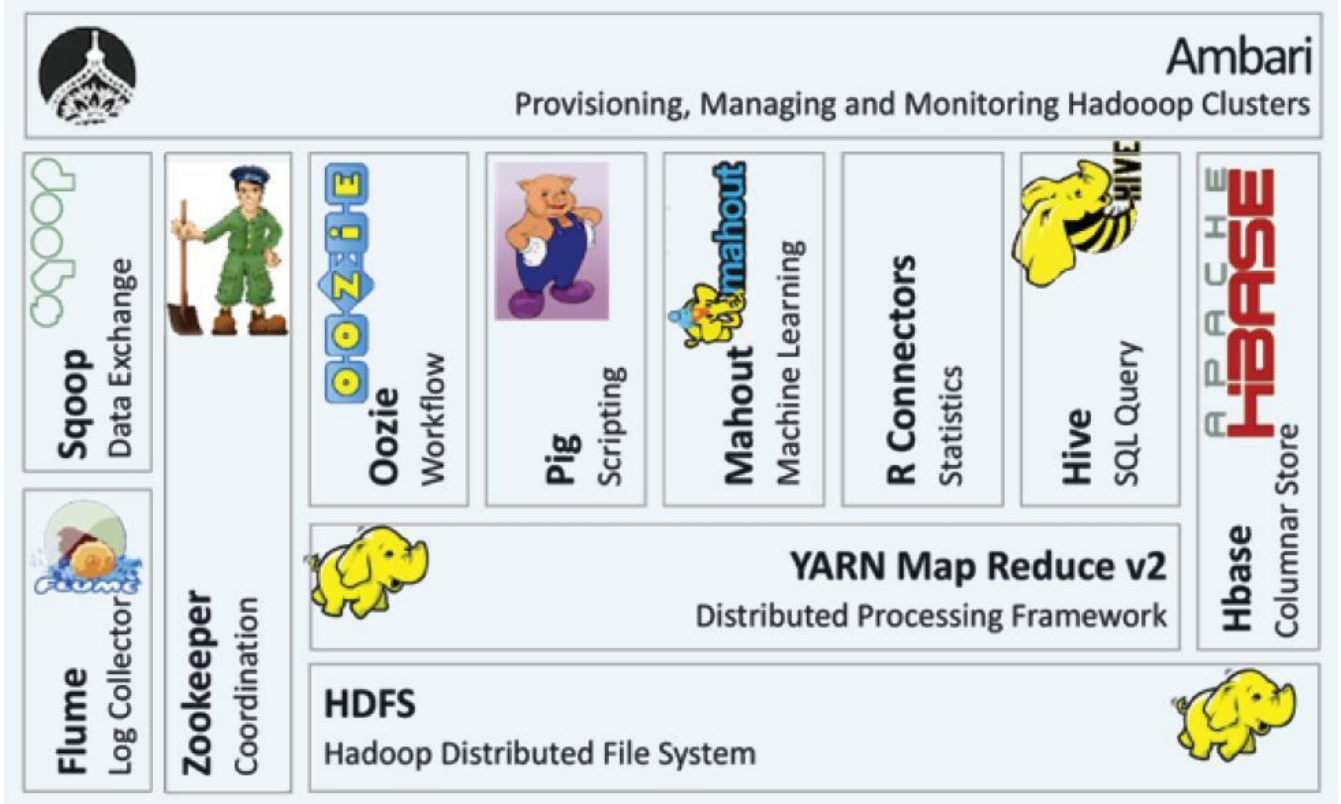
Распределенное хранилище: HDFS

- Данные хранятся на нодах фиксированными чанками
- Реплецирование - обычно фактор 3 (устойчивость к отказам + параллелизация обработки)
- Namenode - хранит метаинформацию о том на каких нодах и в каких чанках хранится файл

Распределенное хранилище: HDFS



Экосистема Hadoop



MRJob

- фреймворк для для программирования и запуска MapReduce задач
- поддерживает запуск задач на Hadoop, Amazon EC2 и Google Cloud
- доступна отладка кода на локальном компьютере
- https://github.com/Yelp/mrjob
- https://github.com/a4tunado/lectures-hse-spark

Пример: подсчет числа слов в корпусе

In [ ]:
```
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
%%file wordcount.py
from mrjob.job import MRJob

class MRWordCount(MRJob):

    def mapper(self, _, line):
        words = line.split()
        # TODO: yiled 'words', intermediate_word_count

    def reducer(self, key, values):
        # key == 'words'
        # TODO: yield 'words', total_word_count
```

In [ ]:
```
import wordcount

job = wordcount.MRWordCount(args=['../data/leo.txt'])
with job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        k, v = job.parse_output_line(line)
        print k, v
```

Пример: операции JOIN

```
In [2]: !head ../data/customers.dat
```

```
Alice Bob|not bad|US
Sam Sneed|valued|CA
Jon Sneed|valued|CA
Arnold Wesise|not so good|UK
Henry Bob|not bad|US
Yo Yo Ma|not so good|CA
Jon York|valued|CA
Alex Ball|valued|UK
Jim Davis|not so bad|JA
```

```
In [1]: !head ../data/countries.dat
```

```
United States|US
Canada|CA
United Kingdom|UK
Italy|IT
```

Пример: операции JOIN

In [ ]:
```
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
%%file mrjoin.py
import os
from mrjob.job import MRJob

class MRJoin(MRJob):

    SORT_VALUES = True  # performs secondary sort

    def mapper(self, key, line):
        map_input_file = os.environ['map_input_file']
        map_input_file = map_input_file.split('/')[-1]
        values = line.split('|')
        # TODO: group records by country

    def reducer(self, key, values):
        # TODO: yield country_full_name, candidate_name
```

In [ ]:
```
import mrjoin

job = mrjoin.MRJoin(args=['../data/countries.dat', '../data/customers.dat'])
with job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        k, v = job.parse_output_line(line)
        print v
```

Пример: логистическая регрессия

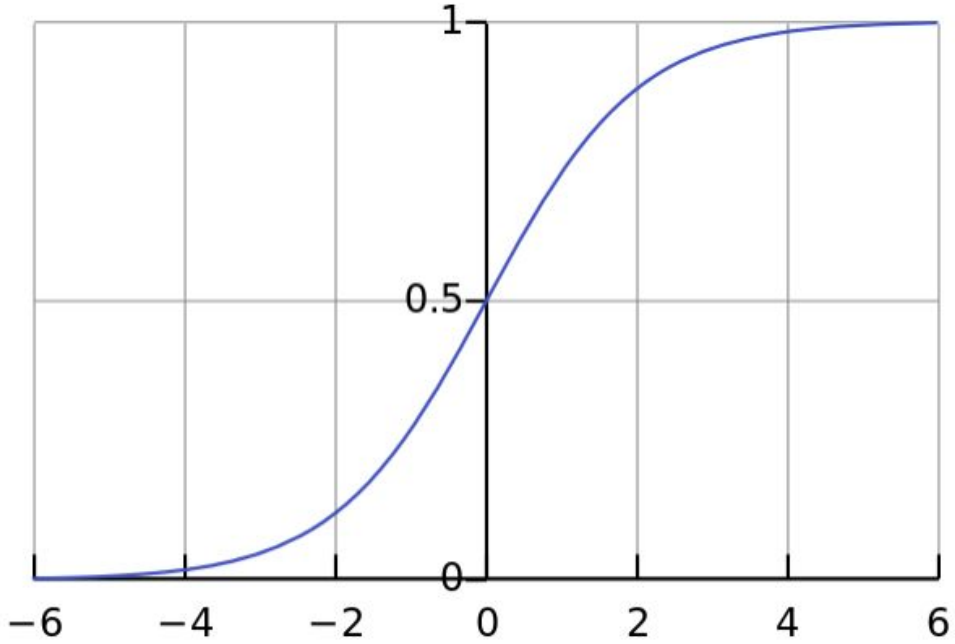
```
In [2]: !head ../data/logit.txt
```

```
1,1.77018953035,0.833701309036
0,0.952262317556,2.3172295198
1,1.72648874607,1.55821601177
0,0.344706947133,3.21632934802
0,-0.138108488444,2.03963087589
0,0.615424887664,2.49574247331
1,1.29912309534,2.13158969782
0,-0.427207932761,2.73523908372
0,0.952018053629,2.34618385185
1,2.84269076004,1.43058836137
```

Пример: логистическая регрессия

$$y(x) = \frac{1}{1 + exp(-xw^T)}$$

$$w = w - \lambda \frac{1}{N} \sum_{i=1}^{N} (y(x_i) - t_i) x_i$$



Пример: логистическая регрессия

In [ ]:
```
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
%%file mrlogit.py
import math
from mrjob.job import MRJob

class MRLogit(MRJob):

    def configure_options(self):
        super(MRLogit, self).configure_options()
        self.add_passthrough_option('--weights', type='str')

    @staticmethod
    def y(x, w):
        # TODO: implement logit function

    def mapper(self, key, line):
        w = map(float, self.options.weights.split(','))
        values = map(float, line.split(','))
        t, x = values[0], values[1:]
        x = [1.] + x
        # TODO: return intermediate gradient values

    def reducer(self, key, values):
        # TODO: calc final gradient value
```

Полезные материалы

- MapReduce: Simplified Data Processing on Large Clusters
  https://research.google.com/archive/mapreduce.html
- Data-Intensive Text Processing with MapReduce
  https://lintool.github.io/MapReduceAlgorithms/index.html
- Python MapReduce framework
  https://pythonhosted.org/mrjob/
- Apache Hadoop Ecosystem
  https://www.cloudera.com/products/open-source/apache-hadoop.html