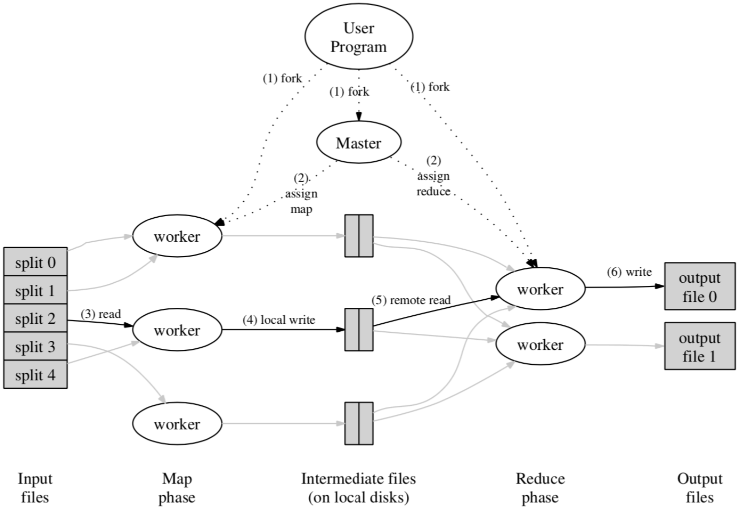
Apache Spark

Ильнур Шугаепов

MapReduce



User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
worker
worker
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Основные приложения MR[1]

- Iterative ML algorithms
- Ad-hoc analytics
- Interactive data-mining

Map Reduce is Good Enough?

[1] Jimmy Lin. "Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail!" In: *Big Data* 1.1 (2013), pp. 28–37.

Ограничения MapReduce

1. Сохраняет (временные) результаты всегда на HDFS
2. Ничего не знает про структуру данных
3. Написание программ из большого числа map,reduce фаз - проблематично

Ограничения MapReduce

Частичные решения

Hive

data wearehousing solution

Pig

dataflow system

Hive[2]

Main components

HiveQL

SQL like query language

Metastore

catalog with metadata about tables

Compiler

converts query to a execution plan

[2] Ashish Thusoo et al. "Hive: a warehousing solution over a map-reduce framework". In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1626–1629.

Pig[3]

Like Hive but with different query language and without Metastore

[3] Alan F Gates et al. "Building a high-level dataflow system on top of Map-Reduce: the Pig experience". In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1414–1425.

Table of Contents

Spark Program

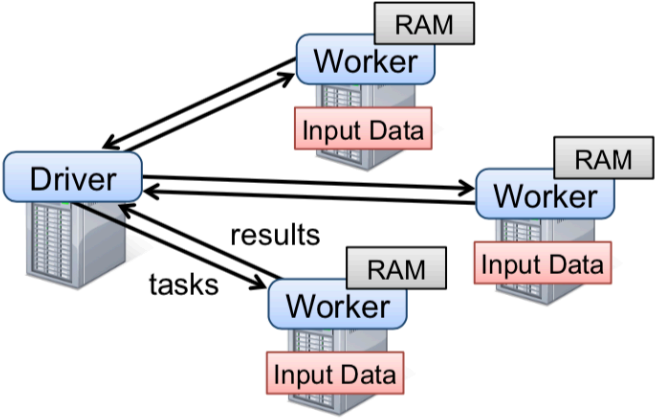


Figure: The user's *driver* program launches multiple *workers*, which read data blocks from a distributed file system

Пример

Поиск по логам

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
errors.persist()

errors.count()

// Count errors mentioning MySQL:
errors.filter(_.contains("MySQL")).count()

// Return the time fields of errors mentioning
// HDFS as an array (assuming time is field
// number 3 in a tab-separated format):
errors.filter(_.contains("HDFS"))
      .map(_.split('\t')(3))
      .collect()
```

Lineage graph

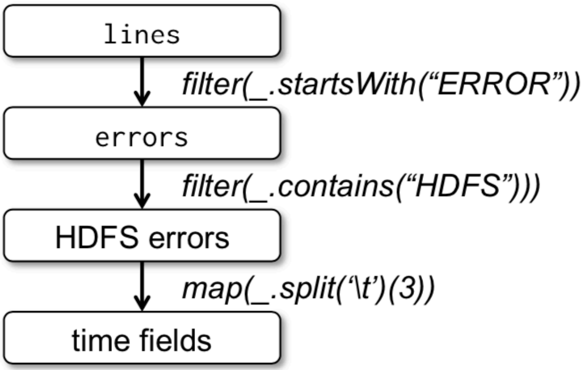


Figure: Boxes represent RDDs and arrows represent transformations

RDD abstraction

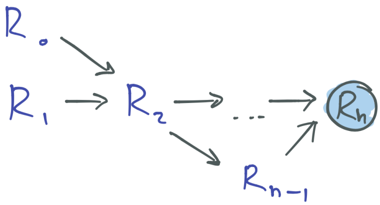
Definition 1 (RDD):

- *Resilient — отказоустойчивый*
- *Distributed — разбитый на партиции*
- *Dataset*

read-only, partitioned collection of records

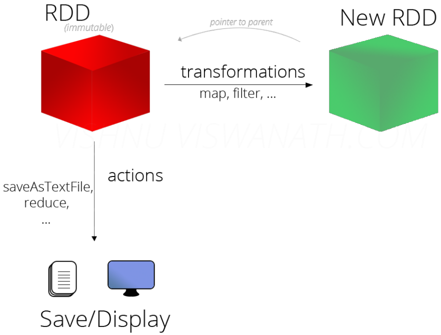
Efficient Fault-tolerance

- Запомним граф вычислений (lineage)
- Тогда если часть данных будет потеряна, то их легко можно восстановить
- RDD знает от каких данных (других RDD) он зависит



Построение RDD

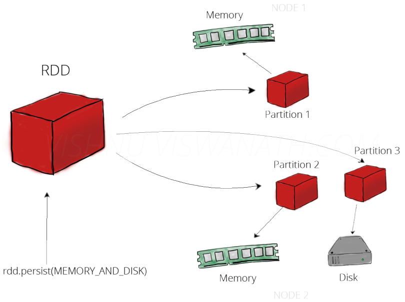
- Из данных находящихся на HDFS или в RAM
- Выполнив операцию над существующим RDD:
  - Transformations
  - Actions



Persistance and Partitioning

- Пользователь может задать каким образом будет храниться RDD

- Пользователь может указать способ партицирования для RDD



Lazy computation

- Spark computes RDDs lazily the first time they are used in an action, so that it can pipeline transformations.
- Spark keeps persistent RDDs in memory by default, but it can spill them to disk if there is not enough RAM.

Основные понятия

Краткий итог

1. Программы на спарке — высокоуровневое описание манипуляций над RDDs.
2. Все вычисления ленивые
3. Пользователь может управлять тем, где будут храниться временные результаты
4. Граф зависимостей обеспечивает высокую надежность вычислений

Table of Contents

Transformations

Types

$$
\begin{aligned}
map(f\colon T \Rightarrow U) &: \quad RDD[T] \Rightarrow RDD[U] \\
filter(f\colon T \Rightarrow Bool) &: \quad RDD[T] \Rightarrow RDD[T] \\
flatMap(f\colon T \Rightarrow Seq[U]) &: \quad RDD[T] \Rightarrow RDD[U] \\
sample(fraction\colon Float) &: \quad RDD[T] \Rightarrow RDD[T] \\
groupByKey() &: \quad RDD[(K, V)] \Rightarrow RDD[(K, Seq[V])] \\
reduceByKey(f\colon (V, V) \Rightarrow V) &: \quad RDD[(K, V)] \Rightarrow RDD[(K, V)] \\
union() &: \quad (RDD[T], RDD[T]) \Rightarrow RDD[T] \\
join() &: \quad (RDD[(K, V)], RDD[(K, W)]) \\
&\qquad \Rightarrow RDD[(K, (V, W))] \\
cogroup() &: \quad (RDD[(K, V)], RDD[(K, W)]) \\
&\qquad \Rightarrow RDD[(K, (Seq[V], Seq[W]))] \\
crossProduct() &: \quad (RDD[T], RDD[U]) \Rightarrow RDD[(T, U)] \\
mapValues(f\colon V \Rightarrow W) &: \quad RDD[(K, V)] \Rightarrow RDD[(K, W)] \\
sort(c\colon Comparator[K]) &: \quad RDD[(K, V)] \Rightarrow RDD[(K, V)] \\
partitionBy(p\colon Partitioner[K]) &: \quad RDD[(K, V)] \Rightarrow RDD[(K, V)]
\end{aligned}
$$

Actions

Types

$$
\begin{aligned}
count() &\quad : \quad RDD[T] \Rightarrow Long \\
collect() &\quad : \quad RDD[T] \Rightarrow Seq[T] \\
reduce(f\colon (T, T) \Rightarrow T) &\quad : \quad RDD[T] \Rightarrow T \\
lookup(k\colon K) &\quad : \quad RDD[(K, V)] \Rightarrow Seq[V] \\
save(path\colon String) &\quad : \quad \text{Outputs RDD to a storage system}
\end{aligned}
$$

Logistic Regression[4]

Code

```
val points = spark.textFile(...)
                  .map(parsePoint).persist()
var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x)))-1)*p.y
  }.reduce((a,b) => a+b)
  w -= gradient
}
```

- $x \in \mathbb{R}^n, \quad \hat{y} = \sigma(\omega \cdot x), \quad \omega \in \mathbb{R}^n$
- $\mathcal{L} = \frac{1}{N} \sum y_i \log \hat{y}_i + (1-y_i) \log(1-\hat{y}_i)$
- $\omega^t \leftarrow \omega^{t-1} - \eta \frac{\partial \mathcal{L}}{\partial \omega}$

[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

Logistic Regression

Performance

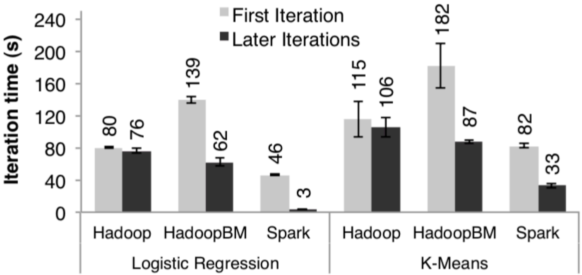


Figure: Duration of the first and later iterations in Hadoop, HadoopBinMem and Spark for logistic regression and k-means using 100 GB of data on a 100-node cluster.

Logistic Regression

Performance

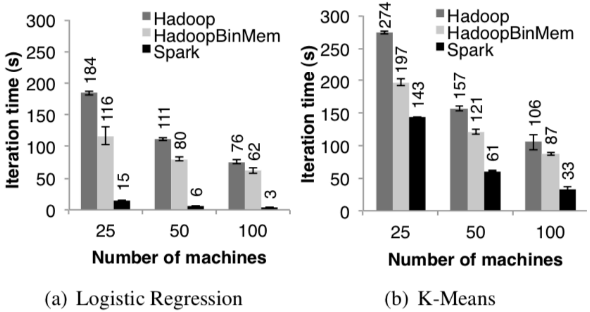


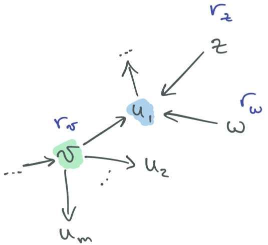
Figure: Running times for iterations after the first in Hadoop, HadoopBinMem, and Spark

Keeping points in memory across iterations can yield a $20\times$ speedup

PageRank[5,6]

Code

```
val links = spark.textFile(...).map(...).persist()
var ranks = // RDD of (URL, rank) pairs
for (i <- 1 to ITERATIONS) {
  // Build an RDD of (targetURL, float) pairs
  // with the contributions sent by each page
  val contribs = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }
  // Sum contributions by URL and get new ranks
  ranks = contribs.reduceByKey((x,y) => x+y)
             .mapValues(sum => a/N + (1-a)*sum)
}
```



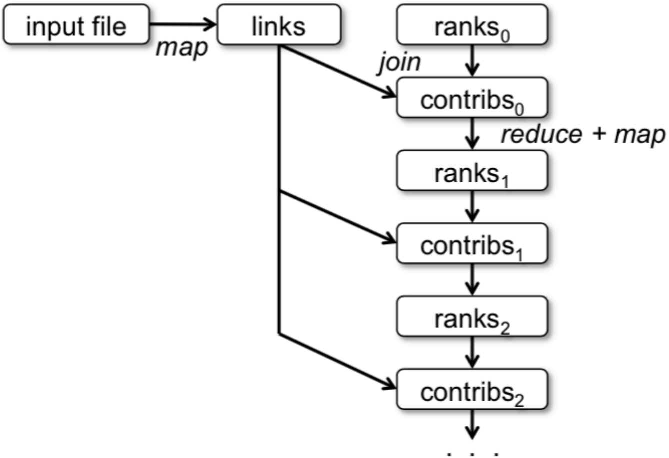
$$r_{u_1}^{(t)} \leftarrow \frac{\alpha}{N} + (1-\alpha) \sum_{v \in In(u_1)} r_v^{(t-1)} / m_v$$

[5] Lawrence Page et al. *The pagerank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999.

[6] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets.* Cambridge university press, 2019.

PageRank

Linage graph



PageRank

Performance

Preserving partitioning might help

```
links = spark.textFile(...).map(...)
              .partitionBy(myPartFunc).persist()
```

If ranks and links are co-partitioned then join requires no communication

PageRank

Performance

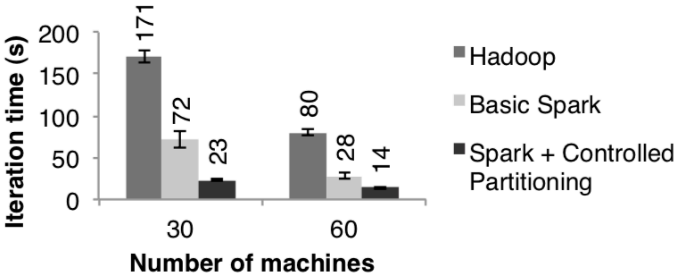


Figure: Performance of PageRank on Hadoop and Spark.

Производительность

Краткий итог

1. Spark работает в 20-100 раз быстрее чем MapReduce
2. Писать программы можно сильно быстрее

Table of Contents

Representing RDDs

Граф

- Partititons — atomic pices of the dataset
- Dependencies — dependencies on parent RDDs

Dependencies

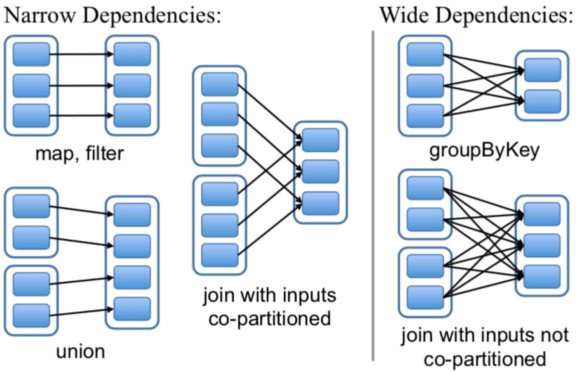


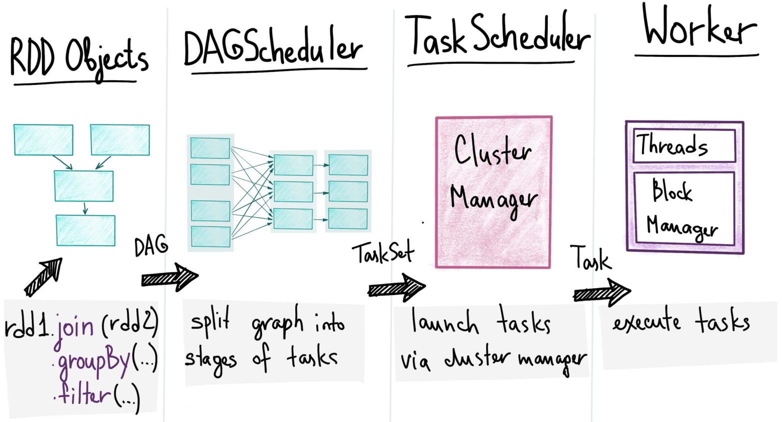
Figure: Examples of narrow and wide dependencies. Each box is an RDD, with partitions shown as shaded rectangles.

Dependencies

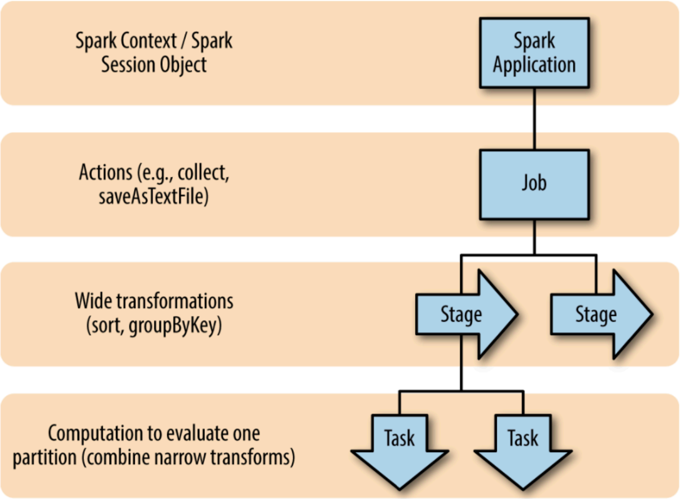
Narrow

- Narrow dependencies allow for pipelined execution on one cluster node
- Recovery after a node failure is more efficient with a narrow dependency

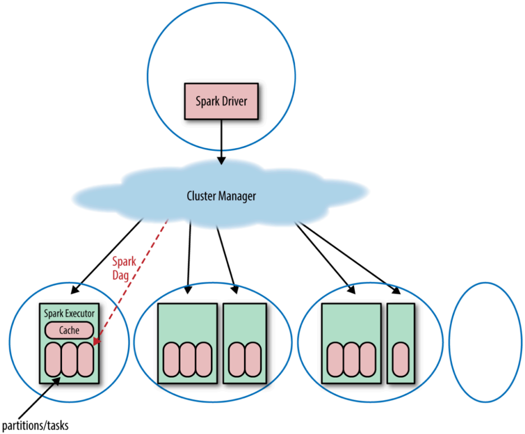
Spark Application Pipeline



Spark Application Tree



Замечания



- One node can have multiple Spark executors, but an executor cannot span multiple nodes.
- An RDD will be evaluated across the executors in partitions (shown as red rectangles).
- Each executor can have multiple partitions, but a partition cannot be spread across multiple executors.

Замечания

SparkContext

Definition 2 (SparkContext): *Connection between user's program and cluster. Containes information about requested resources, type of resources allocation (dynamic/static), etc*

Замечания

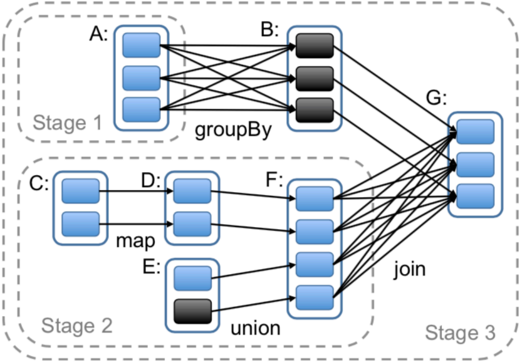


Figure: Boxes with solid outlines are RDDs. Partitions are shaded rectangles, in black if they are already in memory.

Итог

- Gates, Alan F et al. "Building a high-level dataflow system on top of Map-Reduce: the Pig experience". In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1414–1425.
- Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.
- Karau, Holden and Rachel Warren. *High performance Spark: best practices for scaling and optimizing Apache Spark*. " O'Reilly Media, Inc.", 2017.
- Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets*. Cambridge university press, 2019.
- Lin, Jimmy. "Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail!" In: *Big Data* 1.1 (2013), pp. 28–37.
- Page, Lawrence et al. *The pagerank citation ranking: Bringing order to the web*. Tech. rep. Stanford InfoLab, 1999.
- *Spark Overview*. `https://spark.apache.org/docs/latest/index.html`.
- Thusoo, Ashish et al. "Hive: a warehousing solution over a map-reduce framework". In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1626–1629.
- Vavilapalli, Vinod Kumar et al. "Apache hadoop yarn: Yet another resource negotiator". In: *Proceedings of the 4th annual Symposium on Cloud Computing*. 2013, pp. 1–16.

- Zaharia, Matei et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing". In: *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012, pp. 15–28.
- Zaharia, Matei et al. "Spark: Cluster computing with working sets.". In: *HotCloud* 10.10-10 (2010), p. 95.

ml_bd

Вопросы?

Ильнур Шугаепов