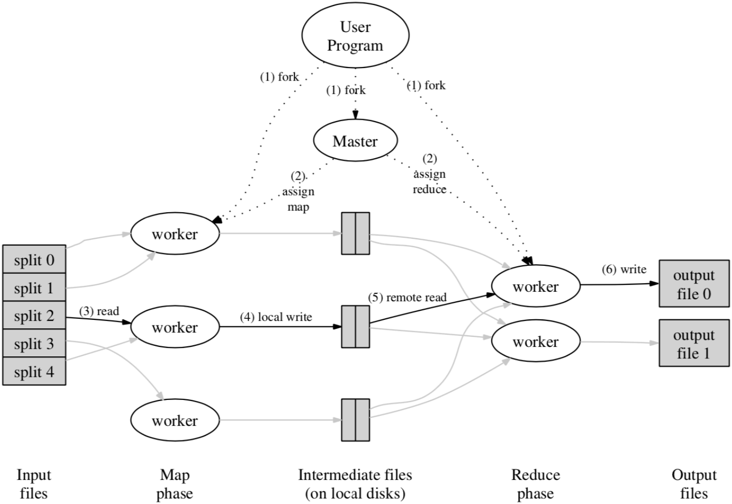
Apache Spark

Шугаепов Ильнур
VK.com
ilnur.shug@gmail.com

январь 2020 г.


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

1. Iterative ML algorithms
2. Ad-hoc analytics
3. Interactive data-mining

Map Reduce is Good Enough?[1]

[1] Jimmy Lin. “Mapreduce is good enough? if all you have is a hammer, throw away everything that’s not a nail!” В: *Big Data* 1.1 (2013), с. 28—37.

- Hive[2] — data wearehousing solution
- Pig[3] — dataflow system

[2] Ashish Thusoo и др. “Hive: a warehousing solution over a map-reduce framework”. В: *Proceedings of the VLDB Endowment* 2.2 (2009), с. 1626—1629.

[3] Alan F Gates и др. “Building a high-level dataflow system on top of Map-Reduce: the Pig experience”. В: *Proceedings of the VLDB Endowment* 2.2 (2009), с. 1414—1425.

Hive
Main components

- HiveQL — SQL like language
- Metastore — catalog with metadata about tables
- Compiler — converts query to a execution plan (MapReduce jobs)

Pig

- Like Hive but with different query language and without Metastore

Определение

RDD:

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

read-only, partitioned collection of records

Efficient Fault-tolerance

- Запомним граф вычислений (linage)
- Тогда если часть данных будет потеряна, то их легко можно восстановить

RDD creation

RDD можно построить одним из следующих способов:

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

Transformations

- map
- filter
- join
- reduceByKey
- ...

Actions

- count — количество элементов в RDD
- save — сохранение RDD, например, на HDFS
- collect — отправить RDD на driver

Persistance and Partitioning

- Users can indicate which RDDs they will reuse and choose a storage strategy for them (e.g., in-memory storage).
- They can also ask that an RDD’s elements be partitioned across machines based on a key in each record.

Операции задают декларативное описание того, что мы хотим сделать

```
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

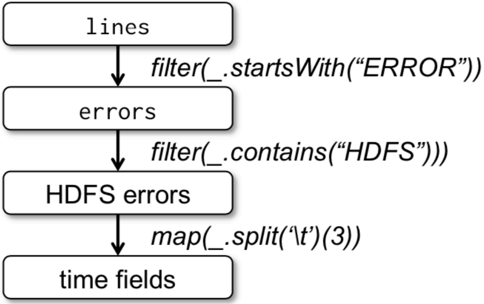


Рис.: Boxes represent RDDs and arrows represent transformations

Linage graph

RDD has enough information about how it was derived from other datasets (its lineage) to compute its partitions from data in stable storage

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.

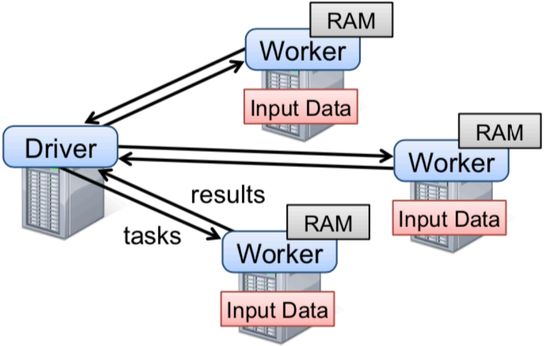


Рис.: The user's *driver* program launches multiple *workers*, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.

Transformations
Types

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

Actions

Types

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

Logistic Regression

LogReg[4]

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

Logistic Regression
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
}
```

Logistic Regression
Performance

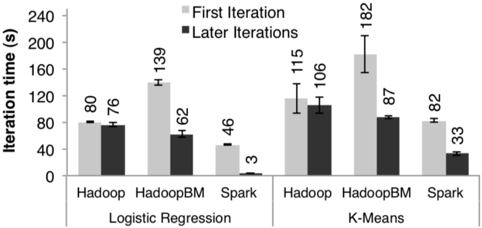


Рис.: Duration of the first and later iterations in Hadoop, HadoopBinMem and Spark for logistic regression and k-means using 100 GB of data on a 100-node cluster.

Logistic Regression
Performance

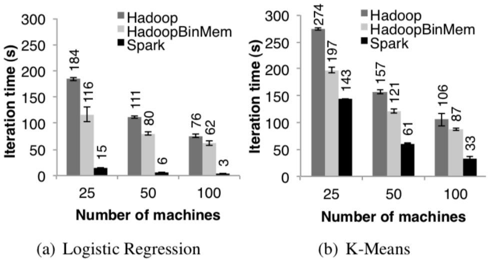


Рис.: Running times for iterations after the first in Hadoop, HadoopBinMem, and Spark

Logistic Regression

Performance

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

PageRank

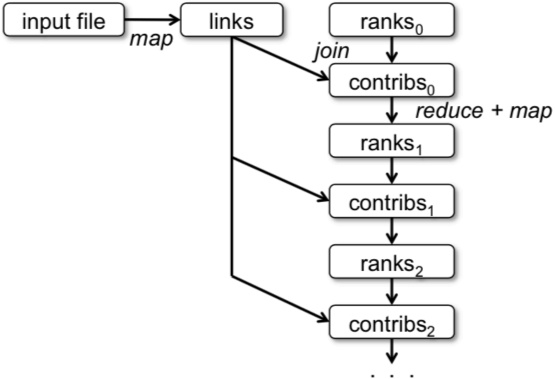
PageRank[5][6]

[5] Lawrence Page и др. *The pagerank citation ranking: Bringing order to the web.* Тех. отч. Stanford InfoLab, 1999.
[6] Jure Leskovec, Anand Rajaraman и Jeffrey David Ullman. *Mining of massive data sets*. Cambridge university press, 2019.

PageRank
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)
}
```

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

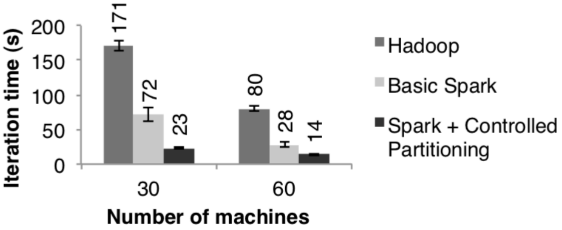


Рис.: Performance of PageRank on Hadoop and Spark.

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

Dependencies

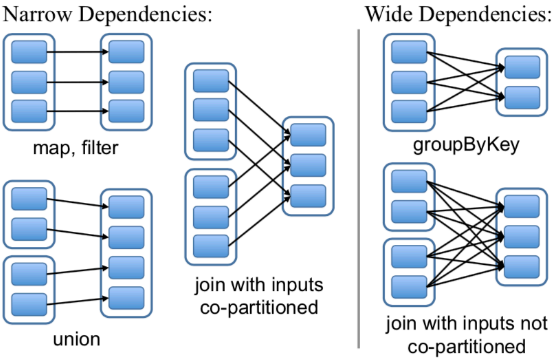


Рис.: Examples of narrow and wide dependencies. Each box is an RDD, with partitions shown as shaded rectangles.

Dependencies

Narrow

- Narrow dependencies allow for pipelined execution on one cluster node, which can compute all the parent partitions.
- Recovery after a node failure is more efficient with a narrow dependency, as only the lost parent partitions need to be recomputed, and they can be recomputed in parallel on different nodes.

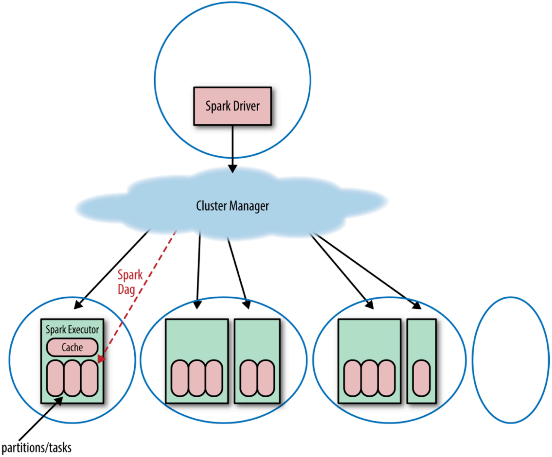


Рис.: Starting a Spark application on a distributed system

Spark Application

Замечание

- *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.*

SparkContext

SparkContext - connection between user’s program and cluster. Containes information about requested resources, type of resources allocation (dynamic/static), etc

Pipeline

1. User runs an action on RDD
2. Scheduler builds a DAG of *stages* (each stage containes piplened transformations with narrow dependencies. *Boundaries* - wide deps.)
3. Scheduler launches *tasks* to compute missing partitions

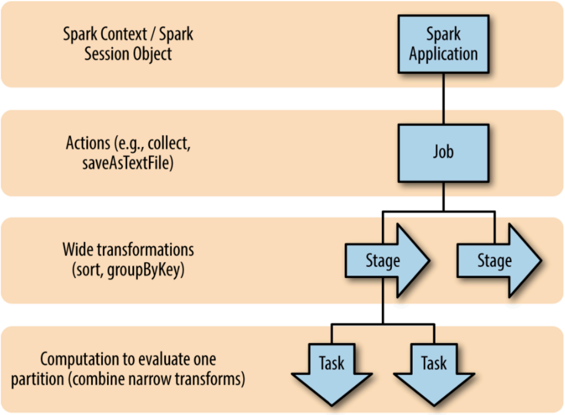


Рис.: The Spark application tree

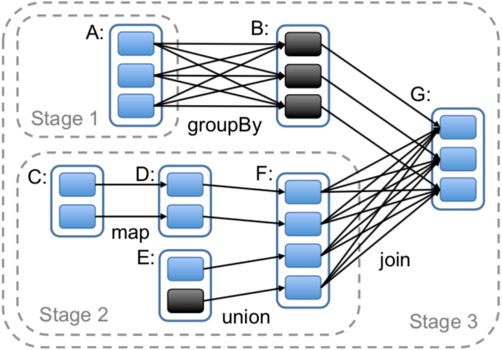


Рис.: Boxes with solid outlines are RDDs. Partitions are shaded rectangles, in black if they are already in memory.

- User can choose how to persist data:
  - in-memory storage as deserialized Java objects
  - in-memory storage as serialized data
  - on-disk storage.
- LRU eviction policy at the level of RDDs

Gates, Alan F и др. “Building a high-level dataflow system on top of Map-Reduce: the Pig experience”. В: *Proceedings of the VLDB Endowment* 2.2 (2009), с. 1414—1425.

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

Karau, Holden и Rachel Warren. *High performance Spark: best practices for scaling and optimizing Apache Spark*. ” O’Reilly Media, Inc.”, 2017.

Leskovec, Jure, Anand Rajaraman и 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!” В: *Big Data* 1.1 (2013), с. 28—37.

Page, Lawrence и др. *The pagerank citation ranking: Bringing order to the web.* Тех. отч. Stanford InfoLab, 1999.

- *Spark Overview*. https://spark.apache.org/docs/latest/index.html.
- Thusoo, Ashish и др. "Hive: a warehousing solution over a map-reduce framework". В: *Proceedings of the VLDB Endowment* 2.2 (2009), с. 1626—1629.
- Vavilapalli, Vinod Kumar и др. "Apache hadoop yarn: Yet another resource negotiator". В: *Proceedings of the 4th annual Symposium on Cloud Computing*. 2013, с. 1—16.
- Zaharia, Matei и др. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing". В: *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 2012, с. 15—28.
- Zaharia, Matei и др. "Spark: Cluster computing with working sets.". В: *HotCloud* 10.10-10 (2010), с. 95.