Wrocław University of Science and Technology

Large Scale Data Processing

Lecture 4 – Spark

dr hab. inż. Tomasz Kajdanowicz, Piotr Bielak, Roman Bartusiak

December 15, 2020

HR EXCELLENCE IN RESEARCH

Overview

Spark

Wrocław University of Science and Technology

Spark

APACHE Spark™

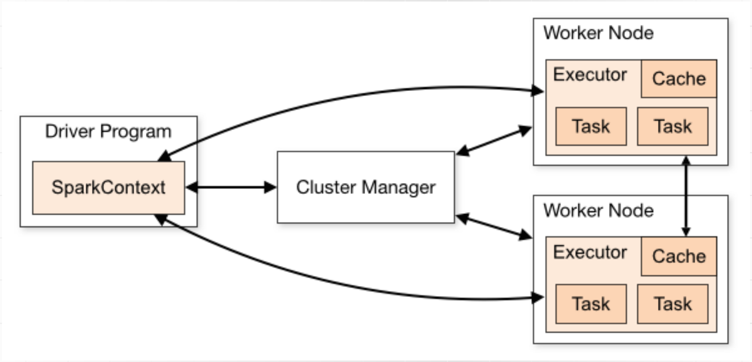
General

Spark

- University of California
- UC Berkeley AMPLab
- Matei Zaharia PhD Thesis
- Huge community
- JVM

Architecture

Spark



Architecture

Spark

- one Driver - many Workers
- Each application in separate JVM
- Driver needs to be accesible from workers

Architecture

Spark

- Appication - our main()
- Driver - executes our main(), schedules DAG
- Executor - each worker spawns executors in order to run tasks of our application

Architecture

Spark

Tune executors per worker

- too small executors - unnecessary overhead
- too big executors - IO issues, failure recovery issues
- keep balance
  - 5 cores per executor?
  - leave core for IO (HDFS, Lustre, ...)
  - leave resources for application manager and other overheads

DAG

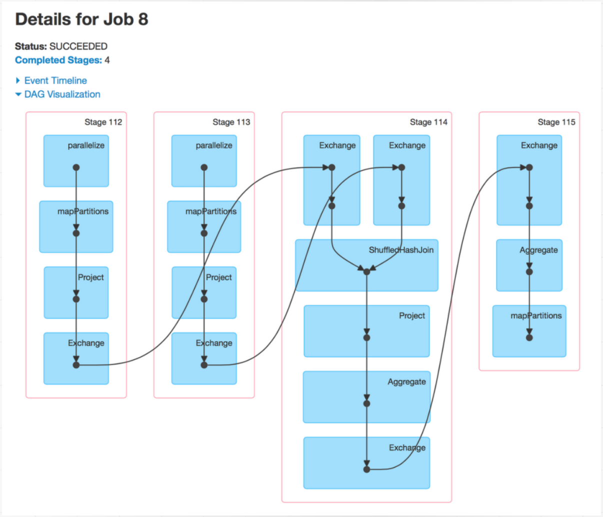
Spark

- Directed Acyclic Graph
- Represents computations
- We are not doing computations in our code, we are creating DAGs and executing them

Wrocław University of Science and Technology

Architecture

Spark



Details for Job 8

Status: SUCCEEDED
Completed Stages: 4

▸ Event Timeline
▾ DAG Visualization

Architecture

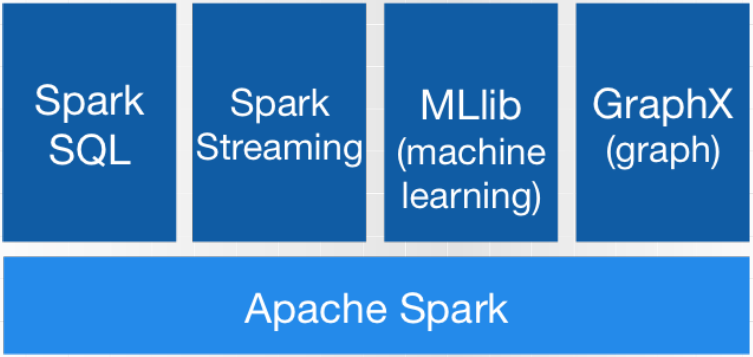
Spark

Application:

- Job
  - Stage (eg. map)
    - Task
    - Task
    - ...
  - Stage (eg. reduce)
  - ...
- Job
- ...

Spark components

Spark



Runtimes

Spark

- Standalone
- YARN
- Mesos
- Kubernetes

Runtimes - Standalone

Spark

Used in WCSS (utilizing pdsdsh)

Simply:

- Put up master
- Take master address
- Put up nodes using master address

Runtimes - YARN

Spark

- Comes from Hadoop
- Primarly only for Hadoop scheduling
- MapReduce V2

Runtimes - MESOS

Spark

- UC Berkeley
- Used by Twitter, AirBnB...
- Full abstraction over resources

Runtimes - Myriad

Spark

- Mesos + YARN on same infrastructure
- YARN running in Mesos

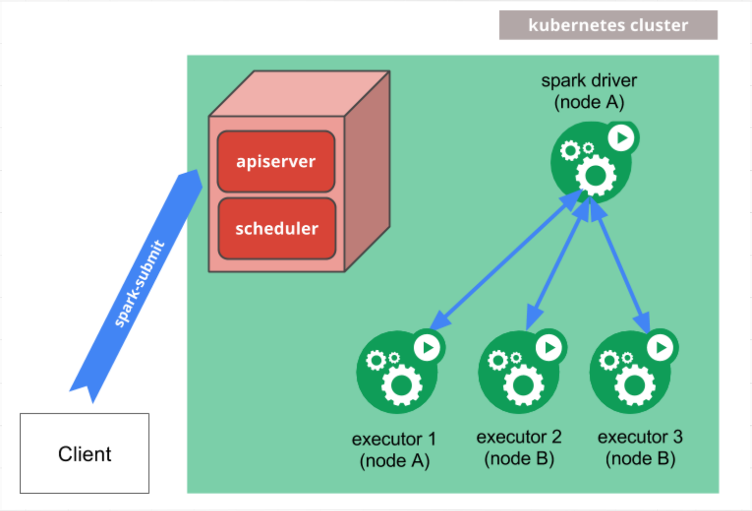
Runtimes - Kubernetes

Spark

- Spark >= 2.3
- Kubernetes >= 1.6

Runtimes - Kubernetes

Spark



Core - primitives

Spark

RDD - resilient distributed dataset

- HDFS
- Hadoop API
- Directly from collections

Wrocław University of Science and Technology

Core - primitives

Spark

Accumulators

- Shared variables between executors
- Only add - efficient

Core - primitives

Spark

Broadcast variables

- Efficient way to distributed read-only data between executors
- Spark optimizes communication in order to minimize overhead
- Reduces overhead when data reused between stages

Core - data partitioning

Spark

- Each RDD is divided into partitions
- Each partition is processed by single executor
- You should have at least equal number of partitions as the number of CPUs in a cluster (taking into account data set size)

Wrocław University of Science and Technology

Core - data partitioning

Spark

- Too big partitions - memory issues
- Too many partitions in comparison to data set size - performance issues
- 2-3 * numCores of partitions (depends on data set size)
- For big data sets - increase the number of partitions

Core - shuffling

Spark

- repartitioning
- expensive
- disk I/O
- network I/O
- serialization/deserialization

Core - data transformations

Spark

Narrow

- Does not require data shuffling
- map, filter ...
- Spark groups narrow transformations - pipelining

Wide

- Requires data shuffling
- groupByKey, ...

Wrocław University of Science and Technology

Core - data transformations

Spark

Remember about load balancing!

- Narrow operations will not cause shuffling
- Without shuffling data can get skewed
- Skewed data -> performance problems
- repartition manually

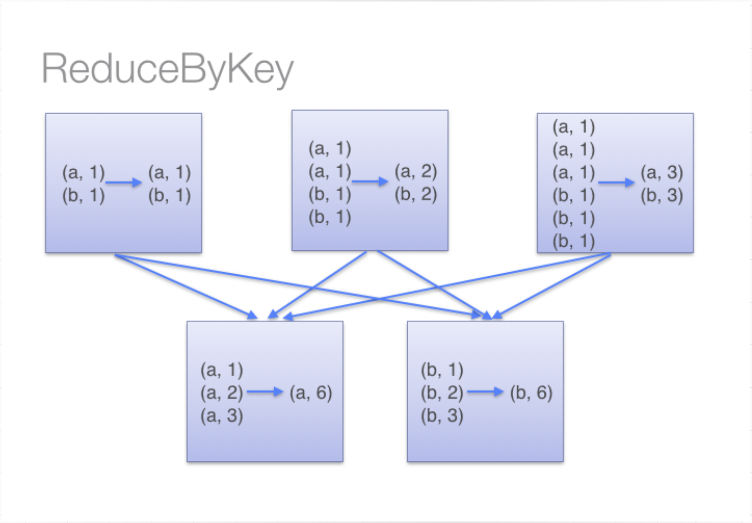
Core - reduceByKey, combineByKey, ... vs groupByKey

reduceByKey, combineByKey, foldByKey decrease data size that needs to be

- saved to disk
- sent over network
- serialized
- deserialized

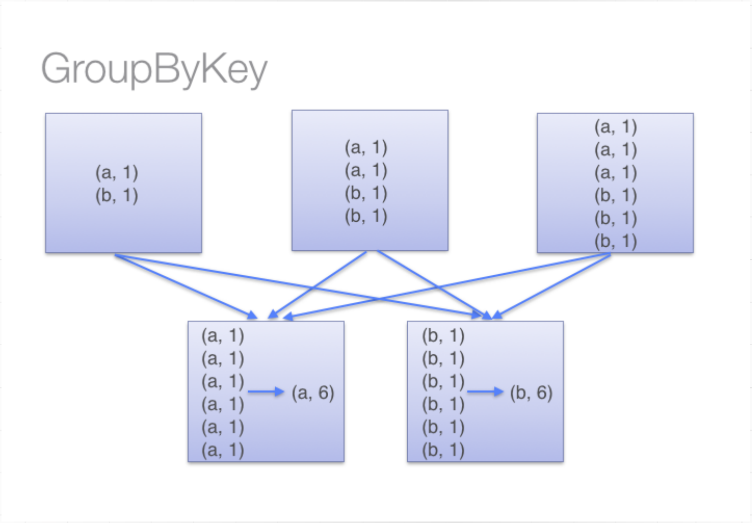
Core - reduceByKey

Spark



Core - groupByKey

Spark



Core - persistence

Spark

- operations are lazy
- we need to persiste operations results in order to reuse it
-

```
rdd.persist()
\\ or
rdd.cache()
```

- can increase performance up to 10x

Core - persistence

Spark

- supports Kryo serialization
- multiple storage levels
- data can be compressed
- off-heap memory support

SparkSQL

Spark

- is a module in Apache Spark that integrates relational processing with Spark's functional programming API.
- lets Spark programmers leverage the benefits of relational processing (e.g., declarative queries and optimized storage), and lets SQL users call complex analytics libraries in Spark (e.g., machine learning)

SparkSQL

Spark

- utilize Spark CORE
- Represents structured and semistructured data
- Use
  - SQL/HiveQL
  - DataSet API

SparkSQL - SQL

Spark

```
// Register the DataFrame as a SQL
//temporary view
df.createOrReplaceTempView("people")

val sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+

```

SparkSQL - DataSet API

Spark

```
case class Person(name: String, age: Long)

// Encoders are created for case classes
val caseClassDS = Seq(Person("Andy", 32))
                  .toDS()
caseClassDS.show()
// +----+---+
// |name|age|
// +----+---+
// |Andy| 32|
// +----+---+

```

SparkSQL - DataSet API

Spark

```
// Encoders for most common types are automatically
//  provided by importing spark.implicits._
val primitiveDS = Seq(1, 2, 3).toDS()
primitiveDS.map(_ + 1).collect() // Returns: Array
    (2, 3, 4)

```

SparkSQL - DataSet API

Spark

```
// DataFrames can be converted to a Dataset by
// providing a class. Mapping will be done by name
val path = "examples/src/main/resources/people.json"
val peopleDS = spark.read.json(path).as[Person]
peopleDS.show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+

```

SparkSQL - DataSet API

Spark

```
val teenagers = peopleDS.where('age >= 10)
.where('age <= 19)
.select('name).as[String]
teenagers.show
//     +------+
//     | name |
//     +------+
//     |Justin|
//     +------+
```

SparkSQL - DataSet API

Spark

```
val symbol = 'someSymbol
// symbol: Symbol = 'someSymbol'

```

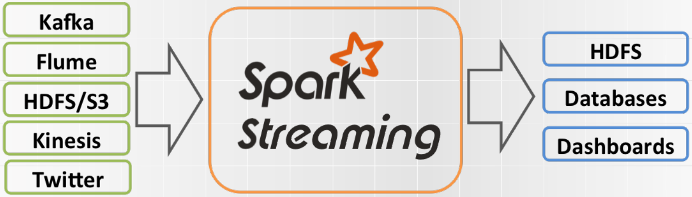
Streaming

Spark

- API mix
- Structured Streaming
- Spark Streaming

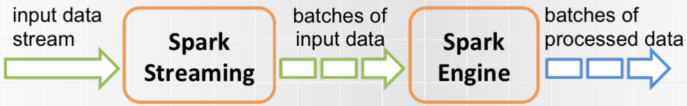
Streaming - Spark Streaming

Spark



Streaming - Spark Streaming

Spark



Streaming - Spark Streaming

Spark

- micro-batching
- configurable latency’s
- can be exactly once guarantees

Streaming - Spark Streaming

Spark

- at most once
- at least once
- exactly once

Streaming - Structured Streaming

Spark

- Standard - 100ms, exactly once
- Continous - 1ms, at least once
  - only map-like
  - SQL without aggregations
  - best with Kafka Source/Sink

MLLib

Spark

- API mix
- Features:
  - data loading
  - data processing
  - ml methods
  - ...

MLLib - DataFrames API

Spark

- pipelines
- friendly
- optimizations
- uniform

MLLib

Spark

- classification
  - binary
  - multi-class
  - multi-label
- regression
- clustering
- collaborative filtering
- frequent-pattern mining

MLLib

Spark

- hyper-parameters search
- cross validation
- train-test split

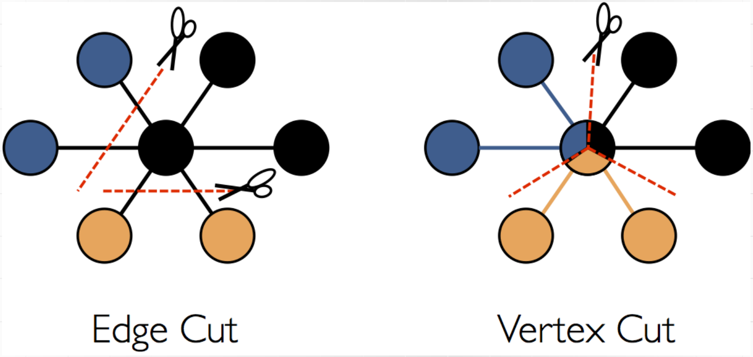
GraphX

Spark

- graph representations on spark
- based on RDD API
- a little of graphs algorithms

GraphX

Spark



Wrocław University of Science and Technology

GraphFrames

Spark

- based on DataFrame API
- schould be faster than GraphX
- smaller API
- in some places, use GraphX under the hood

spark-shell

Spark

- shell for Spark
- created context
- spark API imported

Zeppelin

Spark

- notebooks for Spark
- create, or connect to remote context
- Helium for visualization
- collaboration
- scheduler
- custom dependencies

Spark notebooks

Spark

- more like iPython notebooks
- more built-in visualizations

Spark not for everything

Spark

- Graph structured data
- edge list with data on ’from’ node

Spark not for everything

- Use spark to find *n* level neighbours of node
- find node by name to get identifier *id*
- find neighbours of node *id*
- collect identifiers
- repeat, until get n levels
- collect all data
- query for data objects for each node

Spark not for everything

- Directly call DB using recursive approach
- Do calls in parallel using Cats-effects
- Batch the queries in order to optimize the parallelism
- utilize stream based processing

Wrocław University of Science and Technology

Large Scale Data Processing

Lecture 4 – Spark

dr hab. inż. Tomasz Kajdanowicz, Piotr Bielak, Roman Bartusiak

December 15, 2020