



Wrocław
University
of Science
and Technology

Large Scale Data Processing

Lecture 2 – Data processing, Spark

dr inż. Tomasz Kajdanowicz, Roman Bartusiak, Piotr Bielak

November 9, 2019



HR EXCELLENCE IN RESEARCH

Overview

Paralellism in ML

I/O

POSIX I/O vs HPC

Synchronous processing

Asynchronous processing

Spark

Overview

Paralellism in ML

I/O

POSIX I/O vs HPC

Synchronous processing

Asynchronous processing

Spark

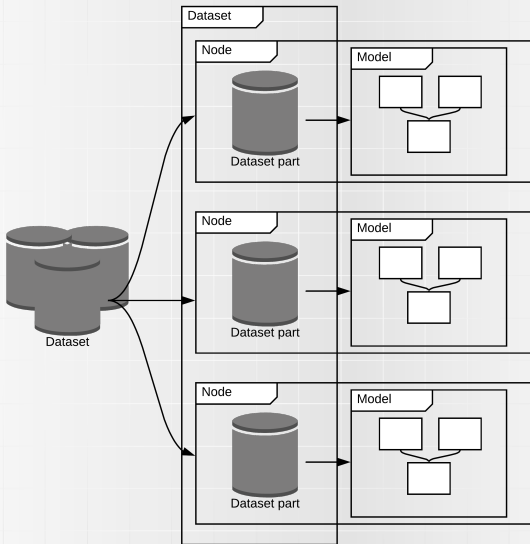
Data Parallelism

Parallelism in ML

- ▶ same model on each distributed node
- ▶ split data among nodes
- ▶ synchronize in order to

Data Parallelism

Paralellism in ML



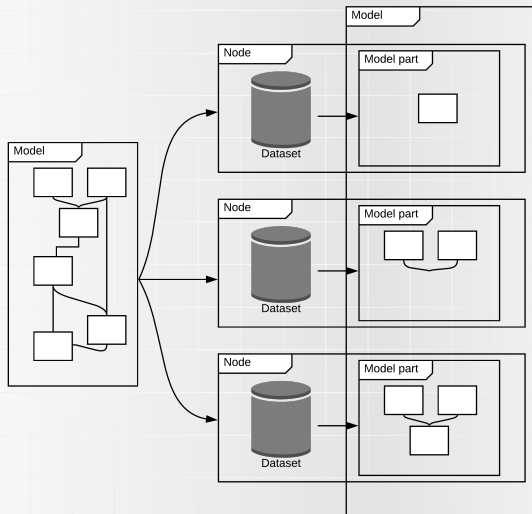
Task Parallelism

Parallelism in ML

- ▶ parts of model on each distributed node
- ▶ same data on each node, or get results of previous part of model

Data Parallelism

Paralellism in ML



Overview

Parallelism in ML

I/O

POSIX I/O vs HPC

Synchronous processing

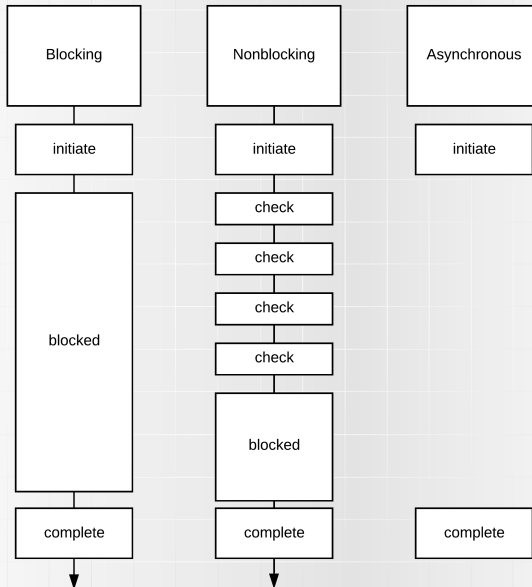
Asynchronous processing

Spark

I/O

- ▶ blocking
- ▶ nonblocking
- ▶ asynchronous

I/O



Overview

Paralellism in ML

I/O

POSIX I/O vs HPC

Synchronous processing

Asynchronous processing

Spark

POSIX I/O vs HPC

- ▶ POSIX is state-full, OS track all file descriptors
- ▶ POSIX gives a lot of unneeded metadata
- ▶ POSIX has strong consistency - after write, you can read it

POSIX I/O vs HPC

- ▶ HPC applications ensures that two process do not write to same file part
- ▶ in HPC, consistency is reduced to smaller subset that whole cluster
- ▶ noatime

Overview

Paralellism in ML

I/O

POSIX I/O vs HPC

Synchronous processing

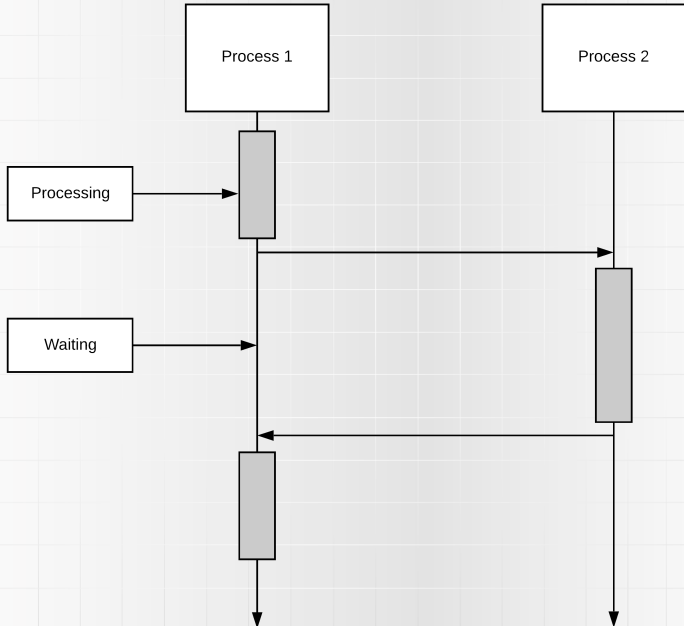
Asynchronous processing

Spark

Synchronous processing

- ▶ make request
- ▶ wait for response
- ▶ continue processing

Synchronous processing



Overview

Parallelism in ML

I/O

POSIX I/O vs HPC

Synchronous processing

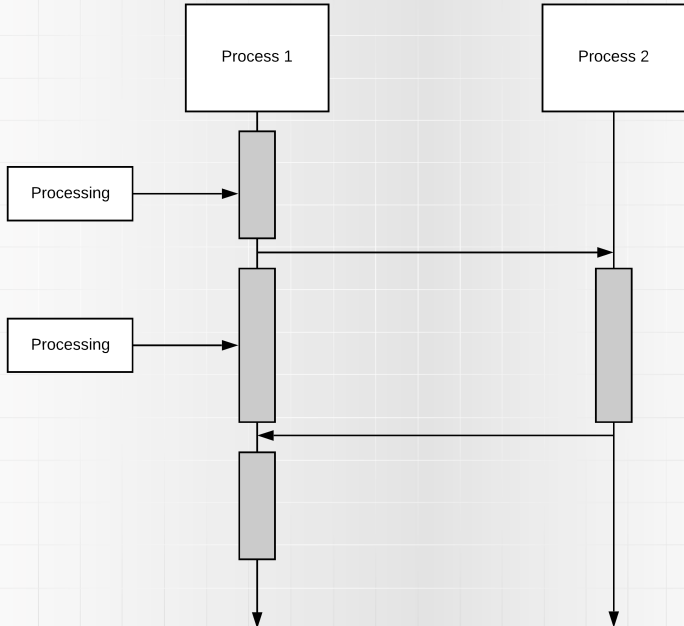
Asynchronous processing

Spark

Asynchronous processing

- ▶ make request
- ▶ continue processing
- ▶ request result arrives, do anything with it
- ▶ continue processing

Asynchronous processing



Mongo - almost async

Asynchronous processing

- ▶ asynchronous client API
- ▶ handles many concurrent connections
- ▶ document level locks
- ▶ what happen when we sand many request through one channel?

Mongo - almost async

Asynchronous processing

- ▶ executed asynchronously?
- ▶ executed in order of arrival, synchronously?

Overview

Parallelism in ML

I/O

POSIX I/O vs HPC

Synchronous processing

Asynchronous processing

Spark



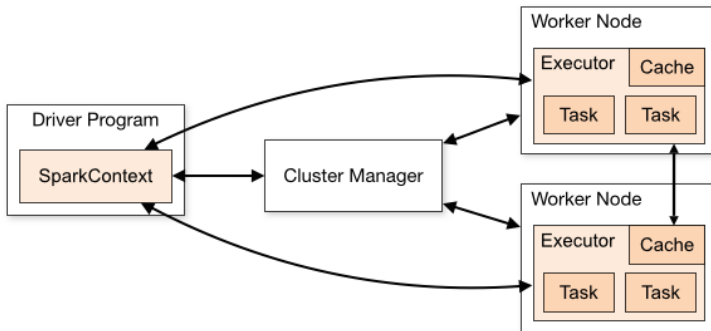
General

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 accessible from workers

Architecture

Spark

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

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

Architecture

Spark

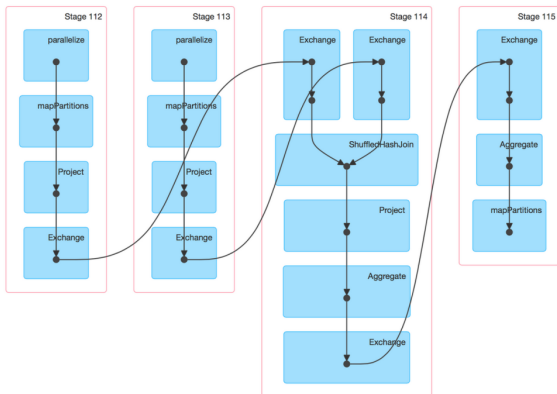
Details for Job 8

Status: SUCCEEDED

Completed Stages: 4

▶ Event Timeline

▼ DAG Visualization



Architecture

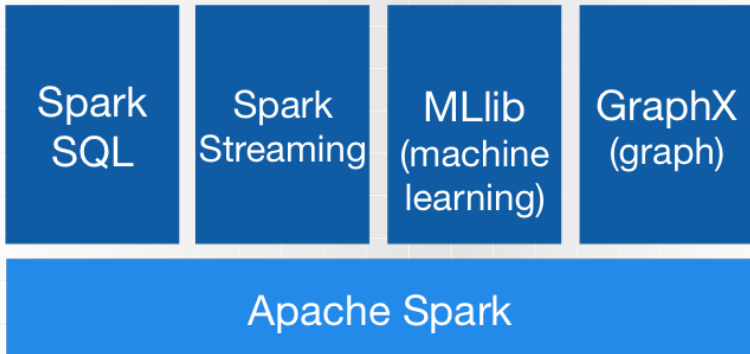
Spark

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
- ▶ Primarily 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 \geq 2.3
- ▶ Kubernetes \geq 1.6

Core - primitives

Spark

RDD - resilient distributed dataset

- ▶ HDFS
- ▶ Hadoop API
- ▶ Directly from collections

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)

Core - data partitioning

Spark

- ▶ Too big partitions - memory issues
- ▶ Too many partitions in comparison to data set size - performance issues
- ▶ $2-3 * \text{numCores}$ 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, ...

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

Spark

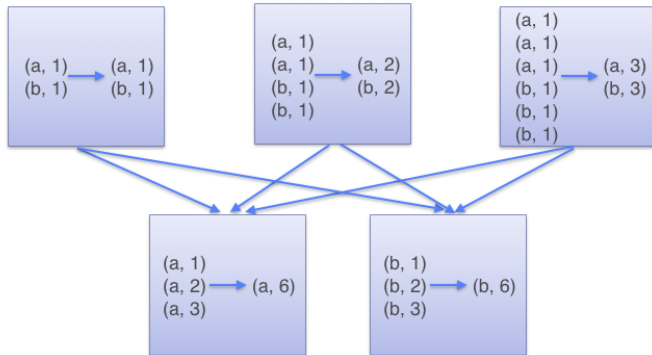
reduceByKey, combineByKey, foldByKey decrease data size that needs to be

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

Core - reduceByKey

Spark

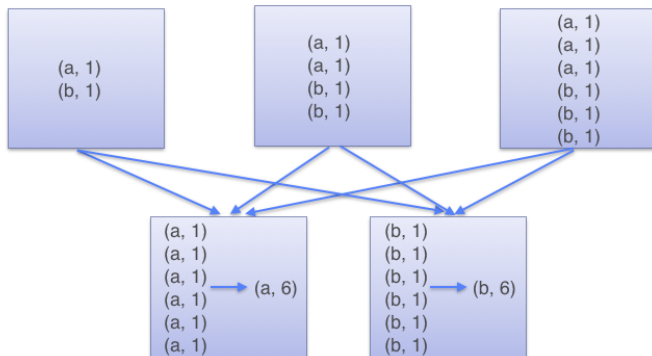
ReduceByKey



Core - groupByKey

Spark

GroupByKey



Core - persistence

Spark

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

```
1 rdd.persist()  
2 \\ or  
3 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

```
1 // Register the DataFrame as a SQL
2 //temporary view
3 df.createOrReplaceTempView("people")
4
5 val sqlDF = spark.sql("SELECT * FROM people")
6 sqlDF.show()
7 // +-----+-----+
8 // | age |    name |
9 // +-----+-----+
10 // | null | Michael |
11 // |   30 |    Andy |
12 // |   19 |   Justin |
13 // +-----+-----+
14
```

SparkSQL - DataSet API

Spark

```
1 case class Person(name: String , age: Long)
2
3 // Encoders are created for case classes
4 val caseClassDS = Seq(Person("Andy", 32))
5                               .toDS()
6 caseClassDS.show()
7 // +-----+-----+
8 // |name|age|
9 // +-----+-----+
10 // |Andy| 32|
11 // +-----+-----+
12
```

SparkSQL - DataSet API

Spark

```
1 // Encoders for most common types are automatically
2 // provided by importing spark.implicits._
3 val primitiveDS = Seq(1, 2, 3).toDS()
4 primitiveDS.map(_ + 1).collect() // Returns: Array
5     (2, 3, 4)
```


SparkSQL - DataSet API

Spark

```
1 // DataFrames can be converted to a Dataset by
2 // providing a class. Mapping will be done by name
3 val path = "examples/src/main/resources/people.json"
4 val peopleDS = spark.read.json(path).as[Person]
5 peopleDS.show()
6 // +---+-----+
7 // | age|   name|
8 // +---+-----+
9 // | null|Michael|
10 // |  30|   Andy|
11 // |  19|  Justin|
12 // +---+-----+
13
```

SparkSQL - DataSet API

Spark

```
1 val teenagers = peopleDS.where('age >= 10)
2   .where('age <= 19)
3   .select('name).as[String]
4 teenagers.show
5 //      +-----+
6 //      |  name  |
7 //      +-----+
8 //      | Justin |
9 //      +-----+
10
```

SparkSQL - DataSet API

Spark

```
1 val symbol = 'someSymbol  
2 // symbol: Symbol = 'someSymbol '  
3
```

Streaming

Spark

- ▶ API mix
- ▶ Structured Streaming
- ▶ Spark Streaming

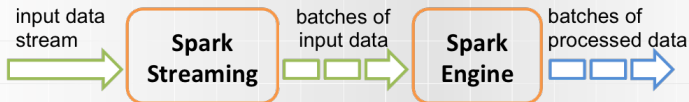
Streaming - Spark Streaming

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
- ▶ Continuous - 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

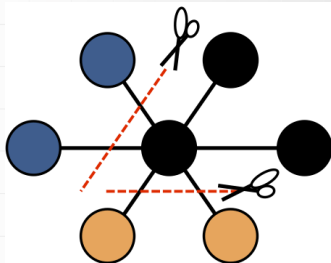
- ▶ classification
 - ▶ binary
 - ▶ multi-class
 - ▶ multi-label
- ▶ regression
- ▶ clustering
- ▶ collaborative filtering
- ▶ frequent-pattern mining

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

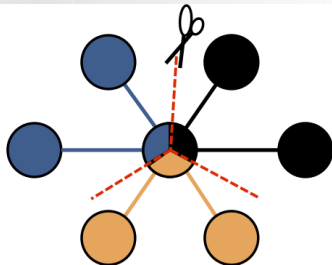
GraphX

Spark

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



Edge Cut



Vertex Cut

GraphFrames

Spark

- ▶ based on DataFrame API
- ▶ should 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

Large Scale Data Processing

Lecture 2 – Data processing, Spark

dr inż. Tomasz Kajdanowicz, Roman Bartusiak, Piotr Bielak

November 9, 2019