Apache Hadoop & Spark – What is it ?

Joseph Allemandou

JoalTech

17 / 05 / 2018

Hello!

Joseph Allemandou

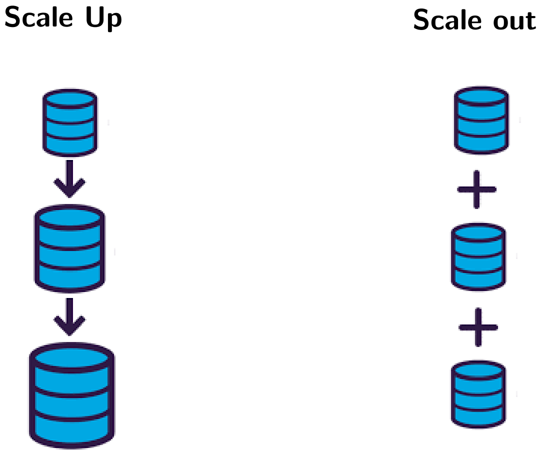
Plan

1. Generalities on High Performance Computing (HPC)

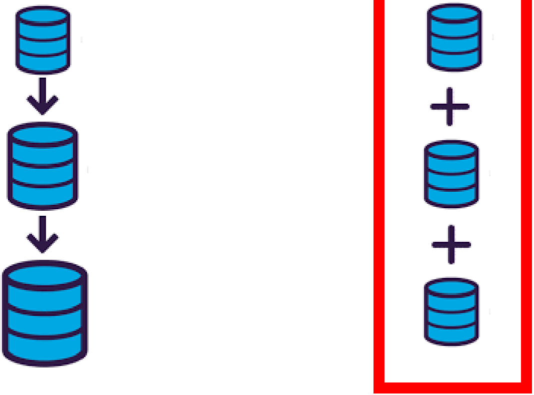
2. Apache Hadoop and Spark – A Glimpse

3. Demonstration

More computation power: Scale up vs. scale out



More computation power: Scale up vs. scale out



Scale Up

Scale out

Parallel computing

Things to consider when doing parallel computing:

- Partitioning (tasks, data)
- Communications
- Synchronization
- Data dependencies
- Load balancing
- Granularity
- IO

Looking back - Since 1950

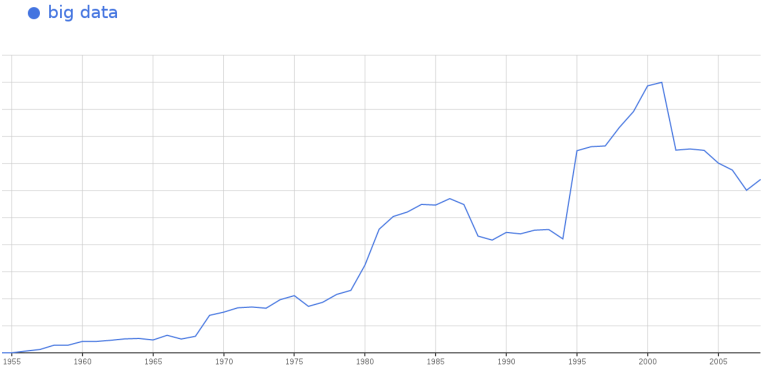


Figure: Google Ngram Viewer

Looking back - Since 1950

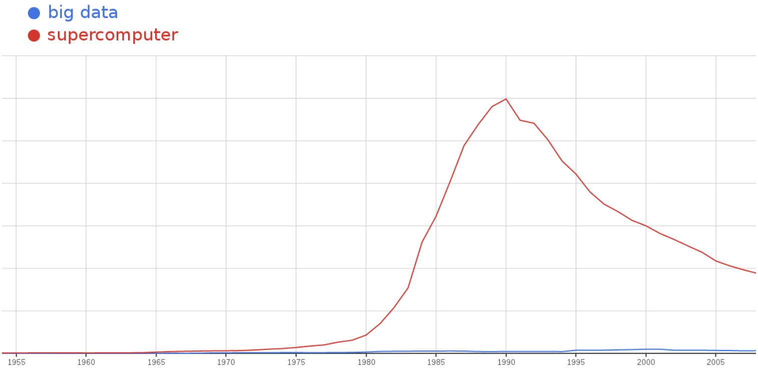


Figure: Google Ngram Viewer

Looking back - Since 1950

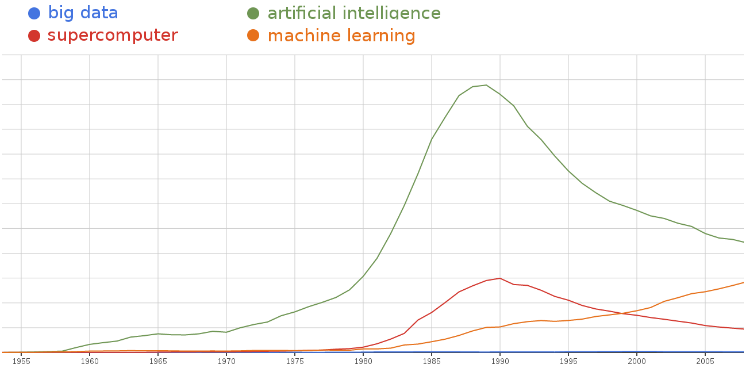


Figure: Google Ngram Viewer

Looking back - Recent times

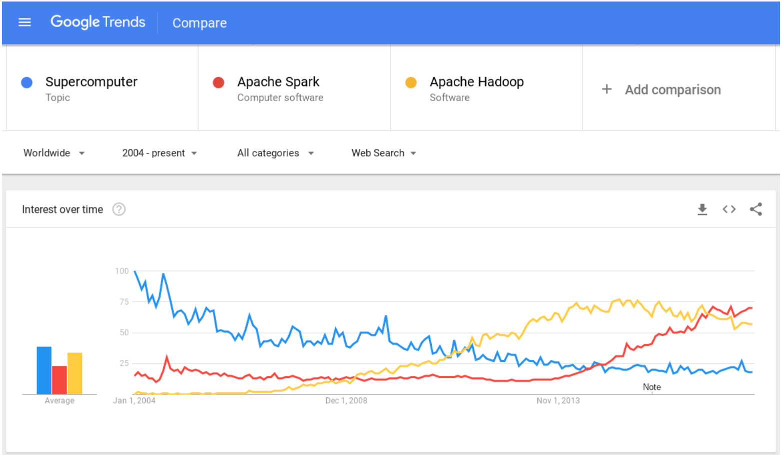


Figure: Google Trends

Same problem, different tools

Supercomputer	**Big Data**
Dedicated hardware	Commodity hardware
MPI	hadoop
Message Passing Interface	APACHE Spark

MPI

- C / C++ / Fortran / Python
- Low-level API - Send / receive messages
- a lot to do manually
  - split the data
  - assign tasks to workers
  - handle synchronisation
  - handle errors

Hadoop + Spark

- Java / Scala / Python / R
- High-level API - dataflows
- Less performant than MPI (see this scientific paper)
- Easier to code than MPI (a lot!)
- High-availability oriented

Plan

1 Generalities on High Performance Computing (HPC)

2 Apache Hadoop and Spark – A Glimpse

3 Demonstration

Apache Hadoop (v2) in (almost) 3 points

Apache Hadoop (v2) in (almost) 3 points

- Distributed storage and computation platform (Java, open source)
  - HDFS – Hadoop Distributed File System
  - YARN – Yet Another Resource Negotiator

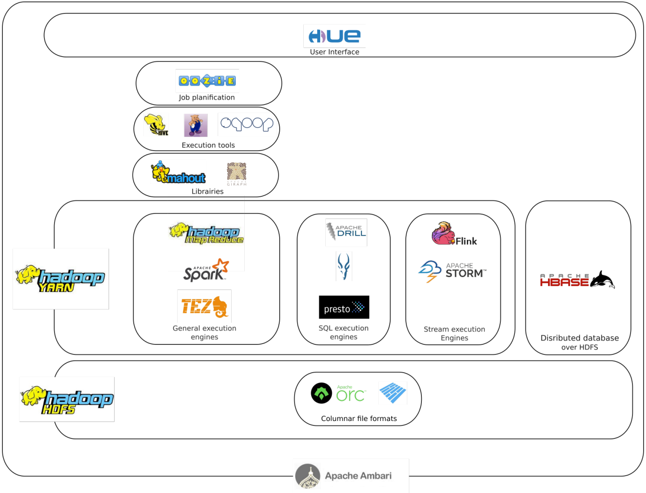
Apache Hadoop (v2) in (almost) 3 points

- Distributed storage and computation platform (Java, open source)
  - HDFS – Hadoop Distributed File System
  - YARN – Yet Another Resource Negotiator
- Built for scalability
  - Up to thousands of nodes (see this tweet)
  - Can be easily extended (heterogeneous hardware)
  - Is resilient to errors (to some extent)

Apache Hadoop (v2) in (almost) 3 points

- Distributed storage and computation platform (Java, open source)
  - HDFS – Hadoop Distributed File System
  - YARN – Yet Another Resource Negotiator
- Built for scalability
  - Up to thousands of nodes (see this tweet)
  - Can be easily extended (heterogeneous hardware)
  - Is resilient to errors (to some extent)
- Working with multiple computation engines
  - Hadoop MapReduce (the good old one)
  - Apache Spark (the new kid on the block)

Apache Hadoop - A big ecosysem



Things to know about Apache Hadoop

Things to know about Apache Hadoop

- HDFS
  - Files are split in blocks – Partitioning !
  - Blocks are replicated – Failure tolerance !
  - Master - Slave architecture (NameNode & DataNodes)

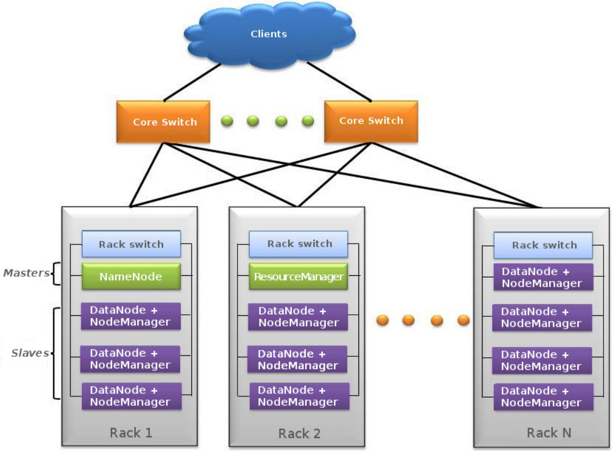
Things to know about Apache Hadoop

- HDFS
  - Files are split in blocks – Partitioning !
  - Blocks are replicated – Failure tolerance !
  - Master - Slave architecture (NameNode & DataNodes)
- YARN
  - Execution containers with defined resources
  - Scheduler manages resource sharing
  - Master - Slave architecture (ResourceManager & NodeManagers)

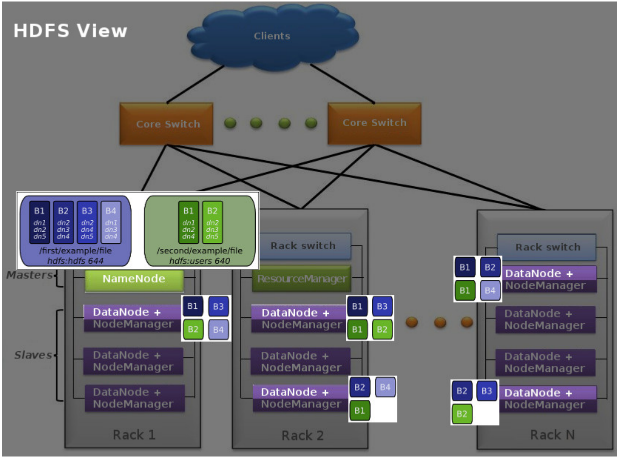
Things to know about Apache Hadoop

- HDFS
  - Files are split in blocks – Partitioning !
  - Blocks are replicated – Failure tolerance !
  - Master - Slave architecture (NameNode & DataNodes)
- YARN
  - Execution containers with defined resources
  - Scheduler manages resource sharing
  - Master - Slave architecture (ResourceManager & NodeManagers)
- Global
  - Execution engines exploit *data locality*

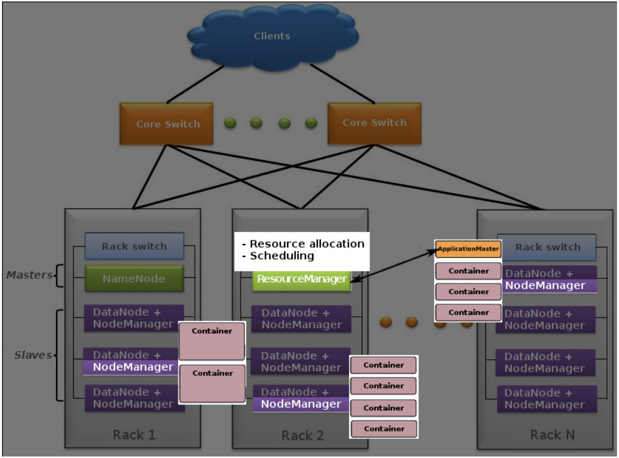
Apache Hadoop - The Frame (HDFS + YARN)



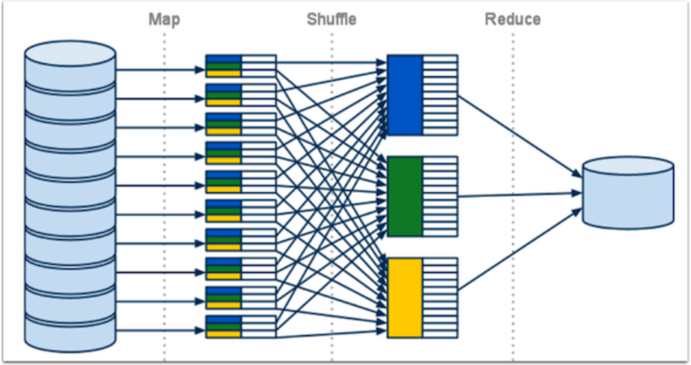
Apache Hadoop - The Frame (HDFS + YARN)



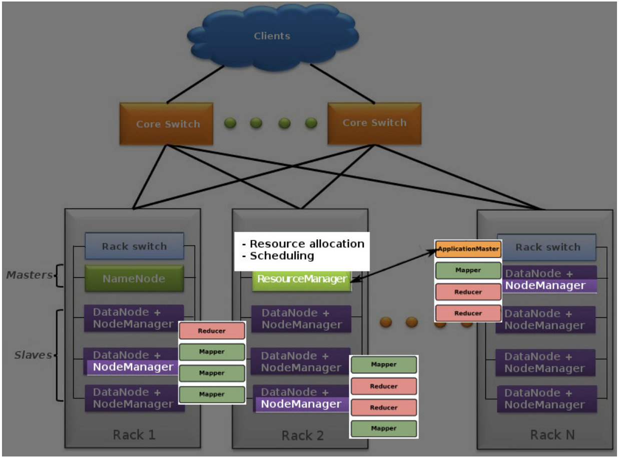
Apache Hadoop - The Frame (HDFS + YARN)



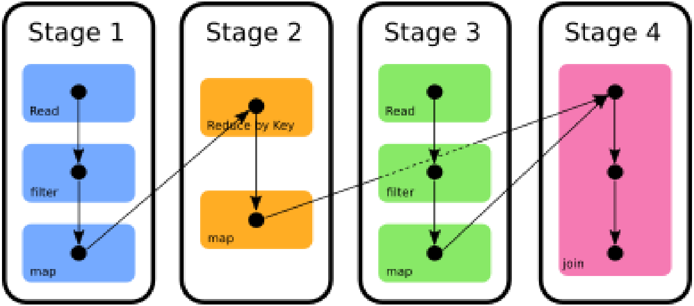
Apache Hadoop - MapReduce



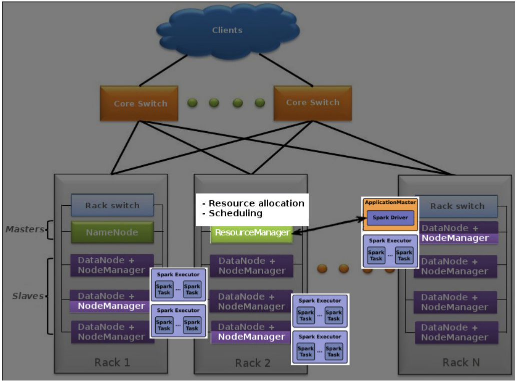
Apache Hadoop - MapReduce



Apache Spark



Apache Spark



Why is Spark preferred to MapReduce?

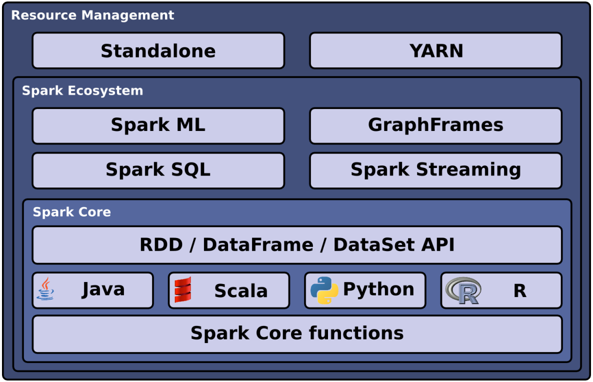
Why is Spark preferred to MapReduce?

- Programming data-flows instead of single map-reduce steps
  - A lot less code for complex flows
  - Data lineage allows for error recovery

Why is Spark preferred to MapReduce?

- Programming data-flows instead of single map-reduce steps
  - A lot less code for complex flows
  - Data lineage allows for error recovery
- Decoupling of tasks and containers – Spark executors run multiple tasks
  - Less executor management overhead
  - Executors can reuse RAM - Caching!

Spark - A big ecosysem as well



Spark - An example of (very) complex flow

Mediawiki History Reconstruction Job

Plan

1 Generalities on High Performance Computing (HPC)

2 Apache Hadoop and Spark – A Glimpse

3 Demonstration

Scala Spark / Toree / Jupyter notebook

Notebook with Spark on Docker

Thank you!