Hálózatba kapcsolt erőforrás platformok és alkalmazásaik

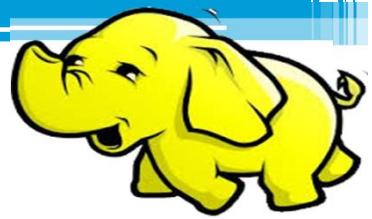
Simon Csaba
TMIT
2019

Google MapReduce -> HADOOP

What is Hadoop?

- Apache top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage
- Java-based framework of tools for storage and large-scale processing of data sets on clusters of hardware
- It is a flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware

What is Hadoop?



- **Hadoop**
  - an open-source software framework that supports data-intensive distributed applications, licensed under the Apache v2 license.
- **Goals / Requirements**
  - Abstract and facilitate the storage and processing of large and/or rapidly growing data sets
    - Structured and non-structured data
    - Simple programming models
  - High scalability and availability
  - Use commodity (cheap!) hardware with little redundancy
  - Fault-tolerance
  - Move computation rather than data

Hadoop's Developers

Doug Cutting

2005: Doug Cutting and Michael J. Cafarella developed Hadoop to support distribution for the Nutch search engine project.

The project was funded by Yahoo.

2006: Yahoo gave the project to Apache Software Foundation.

Google Origins

2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

2004

MapReduce: Simplified Data Processing on Large Clusters

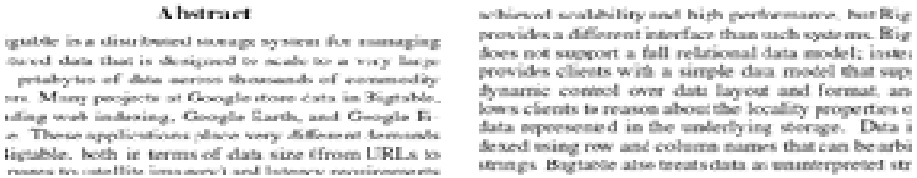
Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com
Google, Inc.

2006

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

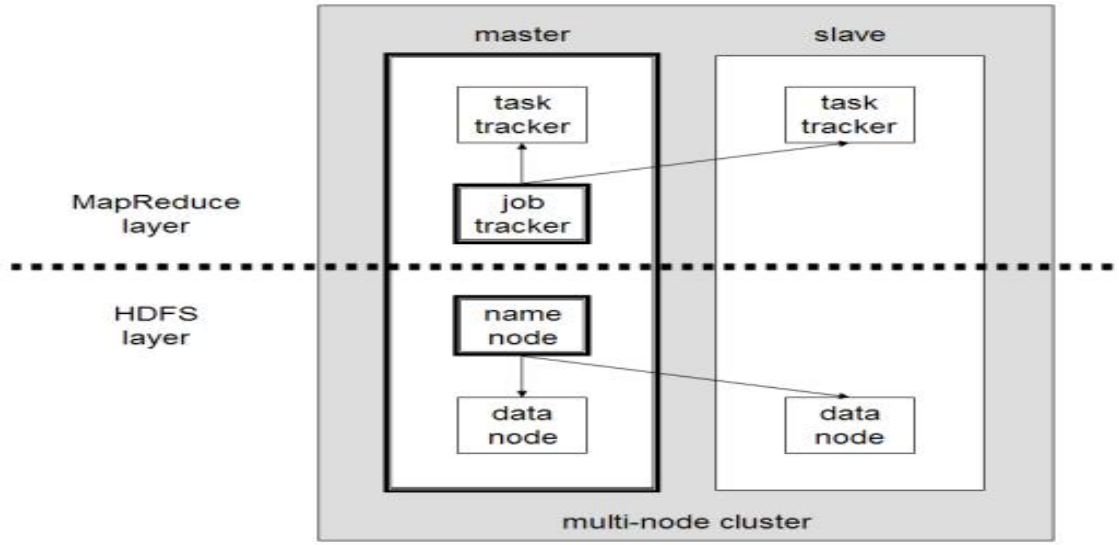
Abstract



HADOOP „Classic”

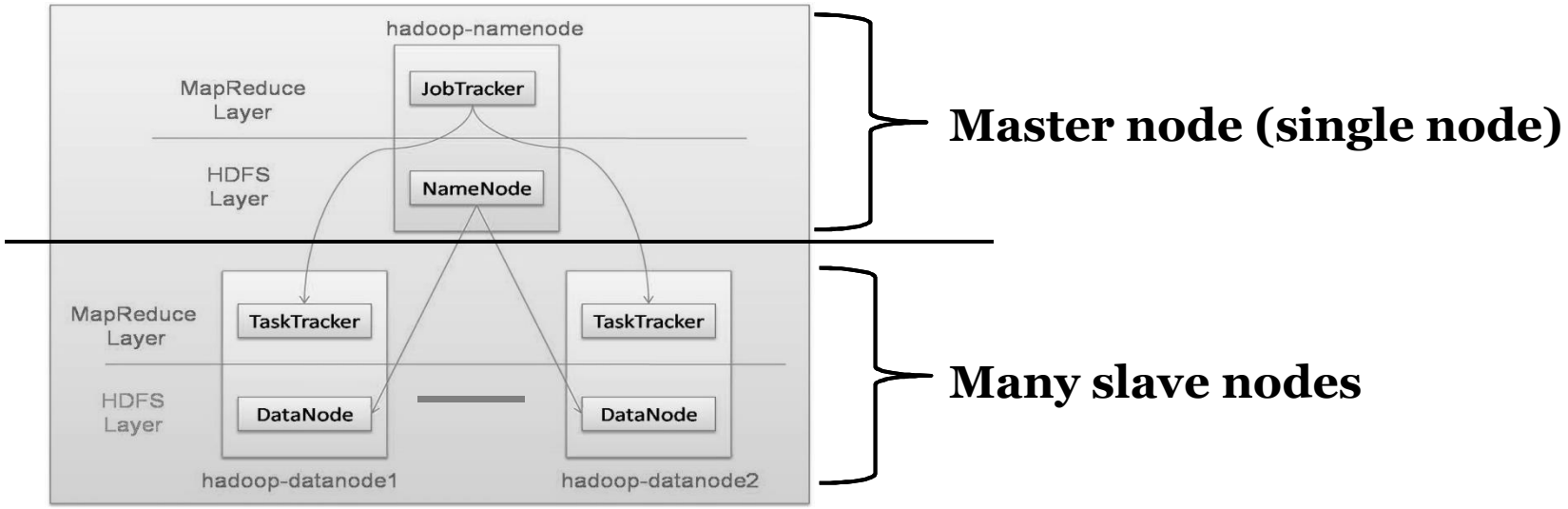
Hadoop framework – the layers

- **Hadoop framework consists on two main layers**
  - Execution engine (MapReduce)
  - Distributed file system (HDFS)

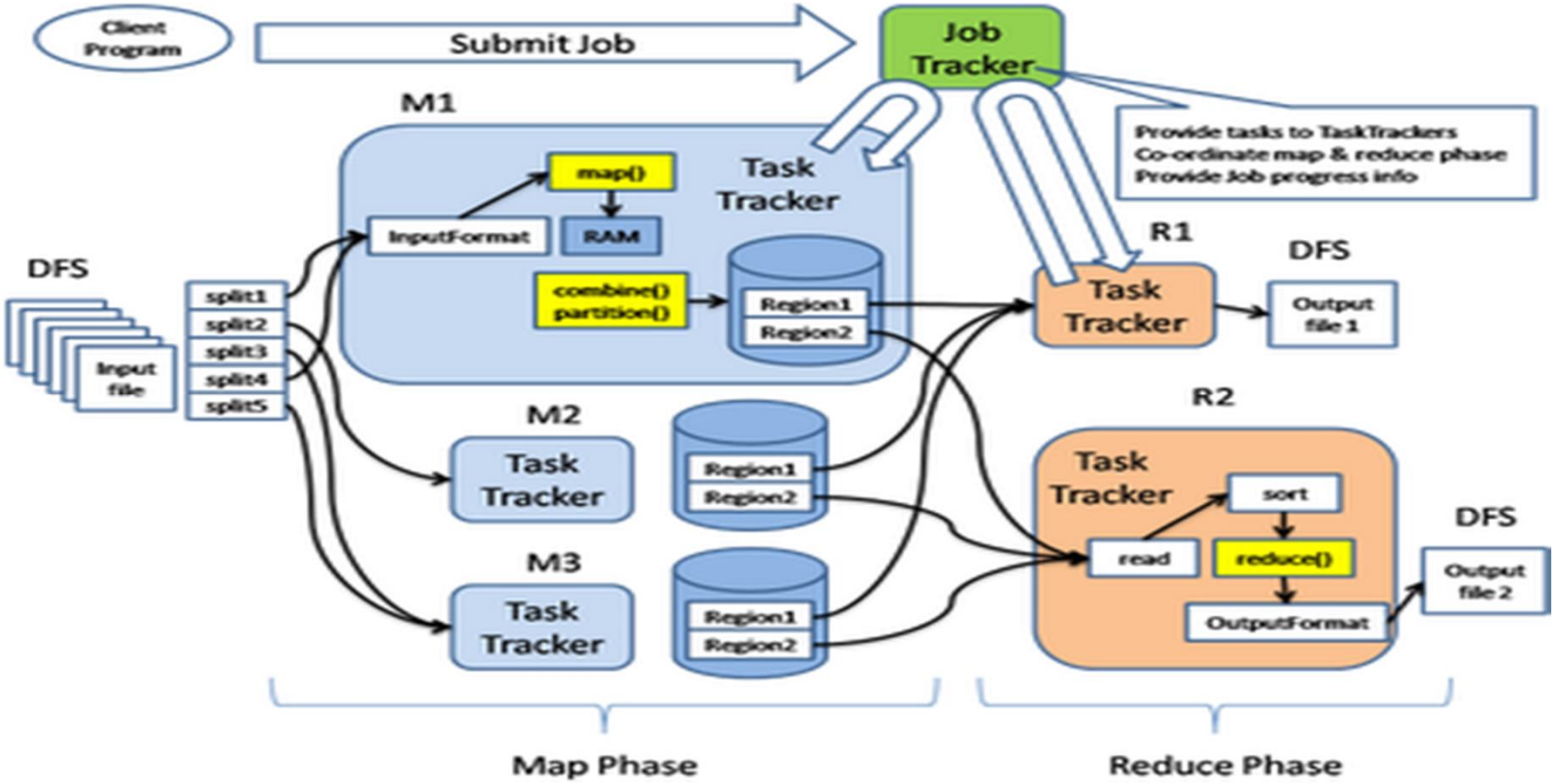


Hadoop Master/Slave Architecture

- Hadoop is designed as a master-slave architecture



Hadoop's Architecture: MapReduce Engine



Hadoop's MapReduce Architecture

MapReduce Engine:

- JobTracker & TaskTracker
- JobTracker splits up data into smaller tasks("Map") and sends it to the TaskTracker process in each node
- TaskTracker reports back to the JobTracker node and reports on job progress, sends data ("Reduce") or requests new jobs

The MapReduce Limitations

- Scalability
  - Maximum Cluster Size – 4000 Nodes
  - Maximum Concurrent Tasks – 40000
  - Coarse synchronization in Job Tracker
- Single point of failure
  - Failure kills all queued and running jobs
  - Jobs need to be resubmitted by users
- Restart is very tricky due to complex state

HDFS

Hadoop's own filesystem

- The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.
- It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant.
  - Highly fault-tolerant and is designed to be deployed on low-cost hardware.
  - Provides high throughput access to application data and is suitable for applications that have large data sets.
  - Relaxes a few POSIX requirements to enable streaming access to file system data.
  - Part of the Apache Hadoop Core project
    http://hadoop.apache.org/core/

MapReduce with HDFS

- Distributed, with some centralization
- Main nodes of cluster are where most of the computational power and storage of the system lies
- Main nodes run TaskTracker to accept and reply to MapReduce tasks, and also DataNode to store needed blocks closely as possible
- Central control node runs NameNode to keep track of HDFS directories & files, and JobTracker to dispatch compute tasks to TaskTracker
- Written in Java, also supports Python and Ruby

HDFS properties

- Hadoop Distributed Filesystem
- Tailored to needs of MapReduce
- Targeted towards many reads of filestreams
- Writes are more costly
- High degree of data replication (3x by default)
- No need for RAID on normal nodes
- Large blocksize (64MB)
- Location awareness of DataNodes in network

HDFS Name Node

- Stores metadata for the files, like the directory structure of a typical FS.
- The server holding the NameNode instance is quite crucial, as there is only one.
- Transaction log for file deletes/adds, etc. Does not use transactions for whole blocks or file-streams, only metadata.
- Handles creation of more replica blocks when necessary after a DataNode failure

HDFS Data Node

- Stores the actual data in HDFS
- Can run on any underlying filesystem (ext3/4, NTFS, etc)
- Notifies NameNode of what blocks it has
- NameNode replicates blocks 2x in local rack, 1x elsewhere

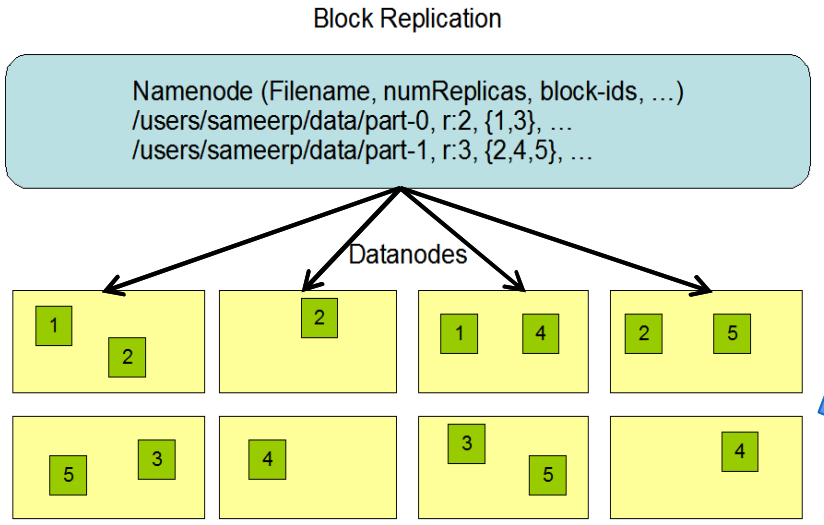
Does Hadoop require HDFS?

- Errr, actually…
- None of these components are necessarily limited to using HDFS
- Many other distributed file-systems with quite different architectures work
- IF Hadoop knows which hosts are closest to the data THEN reduces network traffic
- Many other software packages besides Hadoop's MapReduce platform make use of HDFS

Hadoop: How it Works

Forrás: M. Eltabakh, Hadoop/MapReduce Computing Paradigm

Store the data: HDFS



Centralized namenode

- Maintains metadata info about files

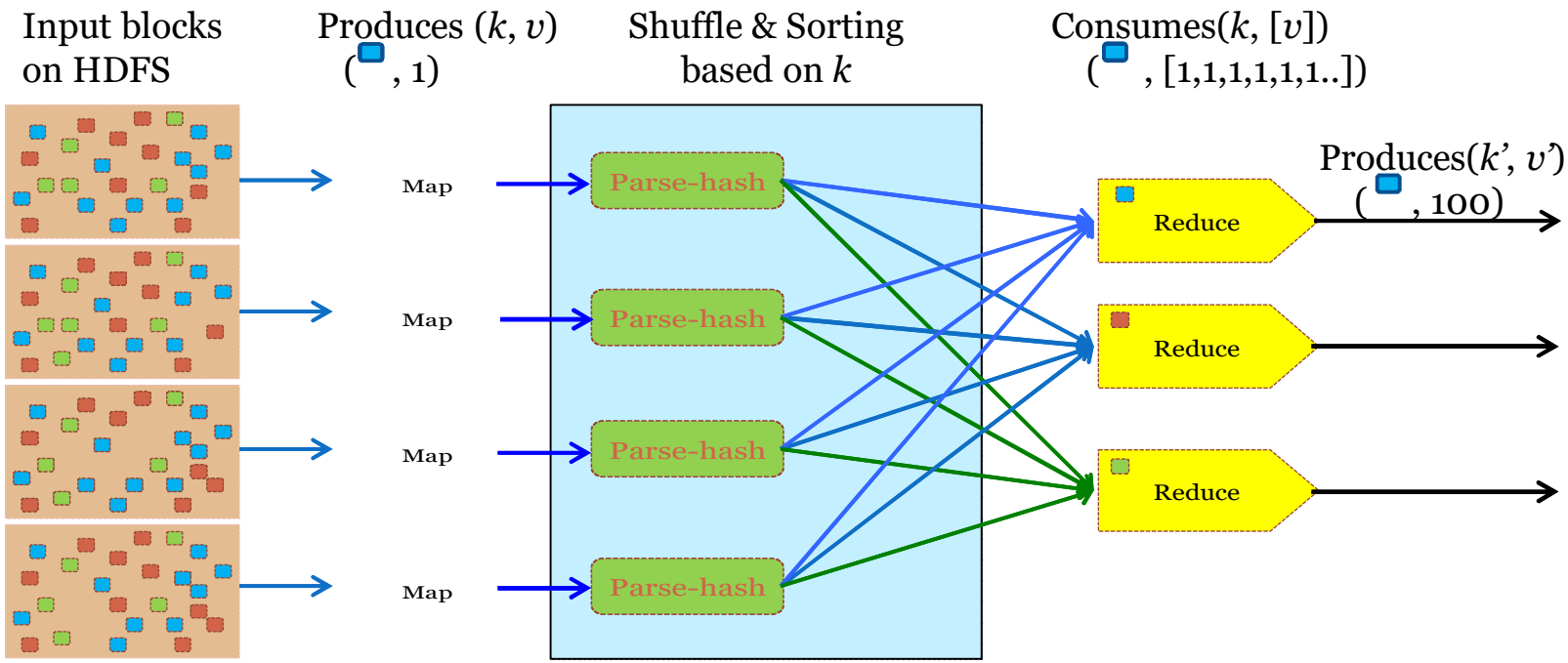
File *F* 1 2 3 4 5

Blocks (64 MB)

Many datanode (1000s)

- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times (Default = 3)

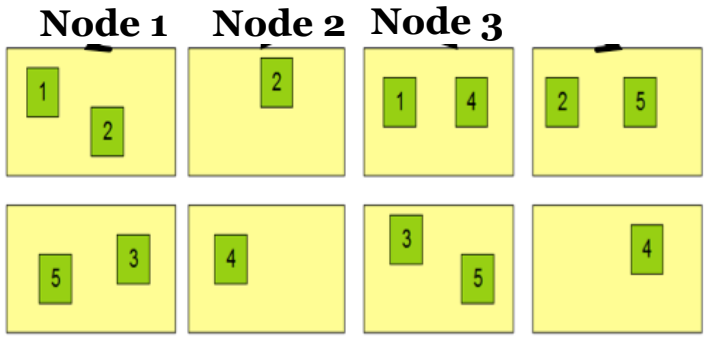
Map-Reduce Execution Engine - Color Count



Users only provide the "Map" and "Reduce" functions

Properties of MapReduce Engine

- **Job Tracker is the master node (runs with the namenode)**
  - Receives the user's job
  - Decides on how many tasks will run (number of mappers)
  - Decides on where to run each mapper (concept of locality)



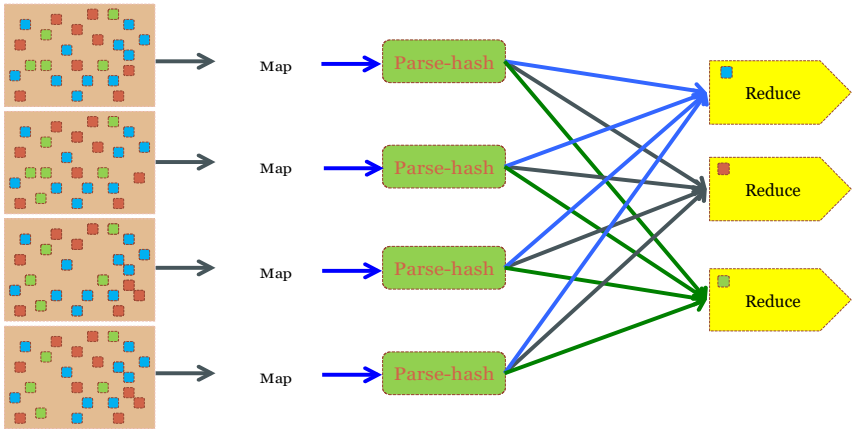
Å This file has 5 Blocks → run 5 map tasks

Å Where to run the task reading block "1"

Å *Try to run it on Node 1 or Node 3*

Properties of MapReduce Engine

- **Task Tracker is the slave node (runs on each datanode)**
  - Receives the task from Job Tracker
  - Runs the task until completion (either map or reduce task)
  - Always in communication with the Job Tracker reporting progress

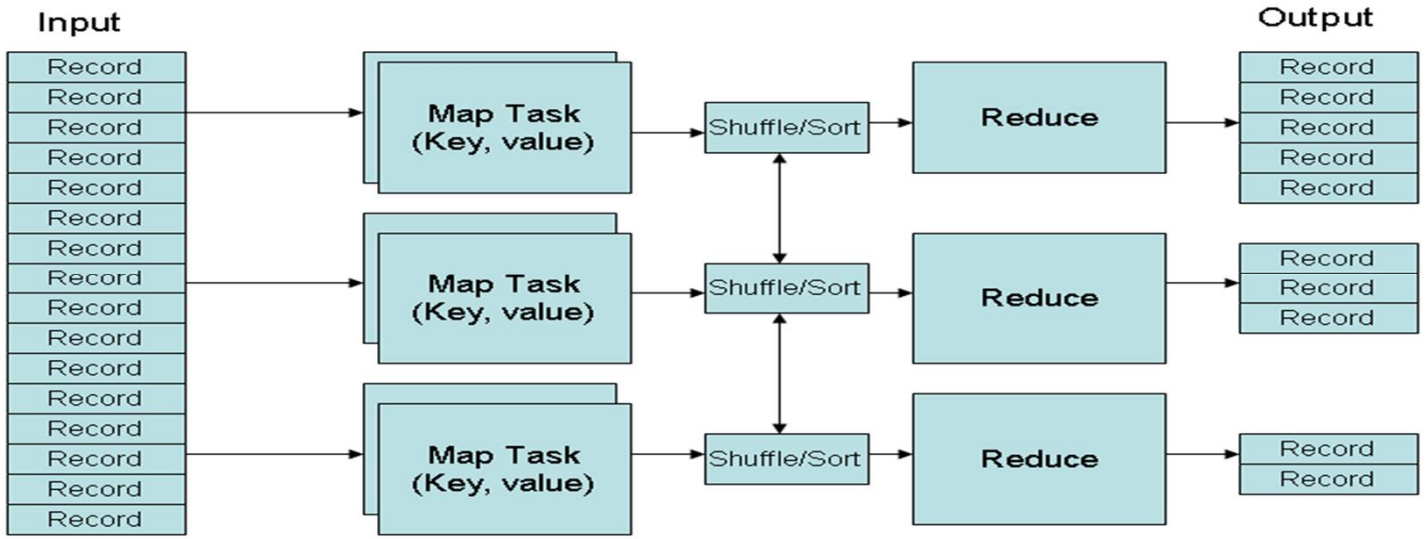


In this example, 1 map-reduce job consists of 4 map tasks and 3 reduce tasks

Key-Value Pairs

- Mappers and Reducers are users' code (provided functions)
- Just need to obey the Key-Value pairs interface
- **Mappers:**
  - Consume <key, value> pairs
  - Produce <key, value> pairs
- **Reducers:**
  - Consume <key, <list of values>>
  - Produce <key, value>
- **Shuffling and Sorting:**
  - Hidden phase between mappers and reducers
  - Groups all similar keys from all mappers, sorts and passes them to a certain reducer in the form of <key, <list of values>>

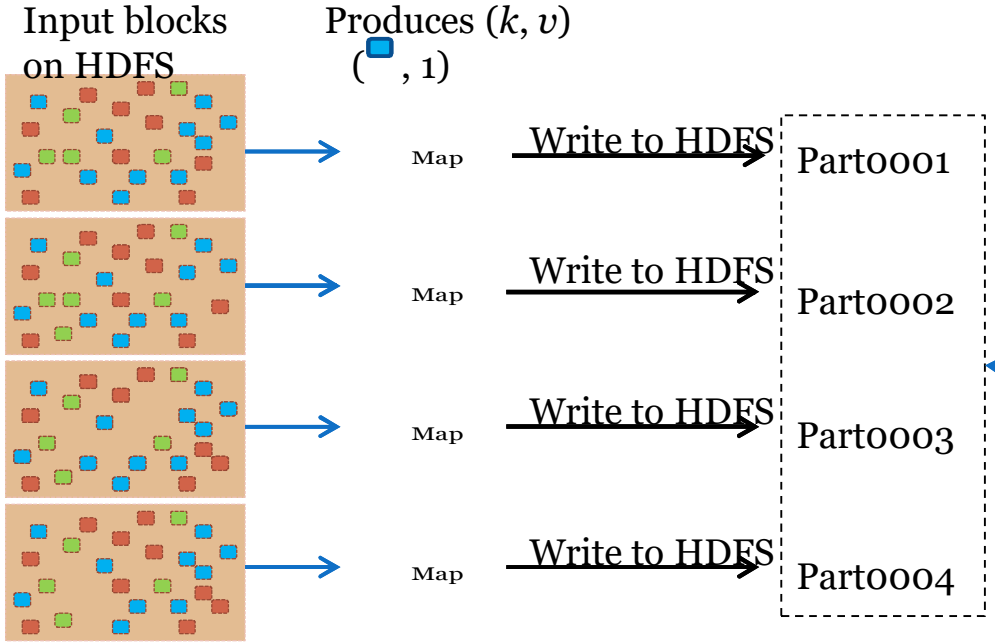
MapReduce Phases



Deciding on what will be the key and what will be the value → developer's responsibility

Color Filter

Job: Select only the blue and the green co



- Each map task will select only the blue or green colors
- No need for reduce phase

That's the output file, it has 4 parts on probably 4 different machines

HADOOP YARN

Hadoop proposal: YARN

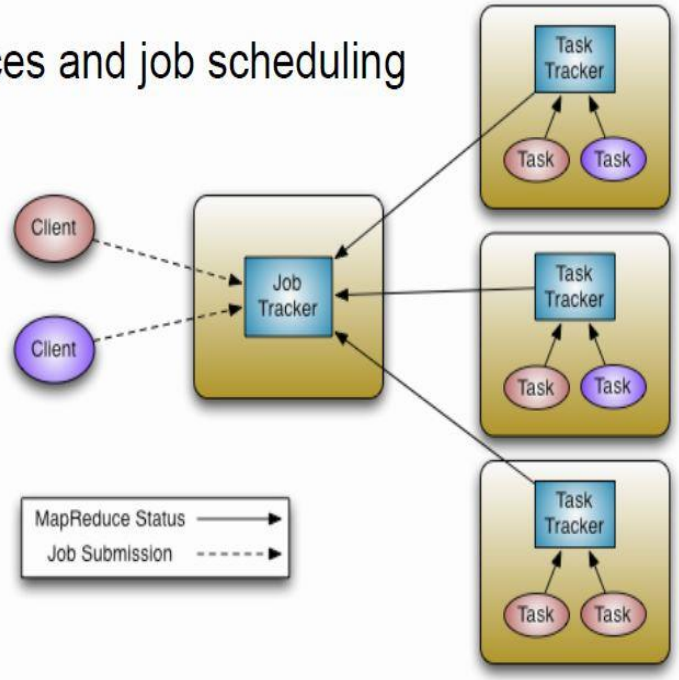
- Yet Another Resource Negotiator
- YARN Application Resource Negotiator(Recursive Acronym)
- Remedies the scalability shortcomings of “classic” MapReduce
- Is more of a general purpose framework of which classic mapreduce is one application.

YARN

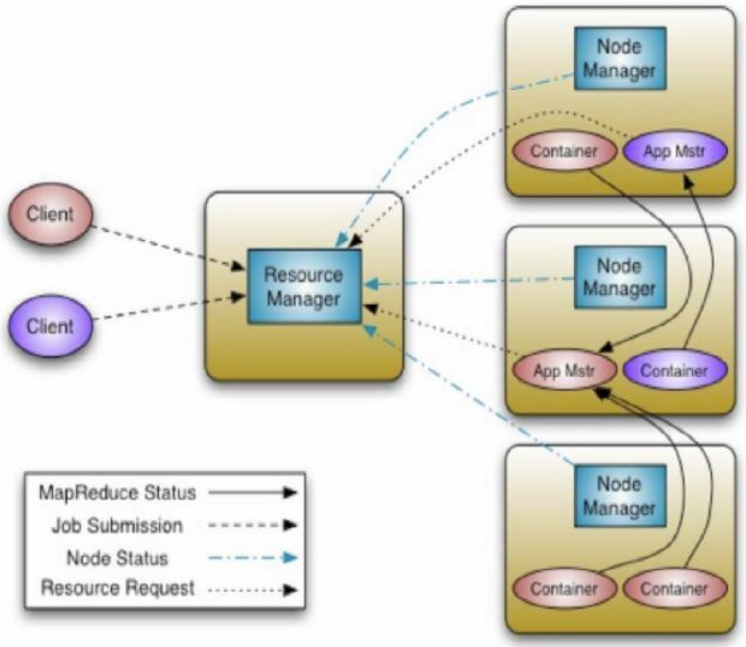
- Split up the two major responsibilities of the JobTracker/TaskTracker into separate entities
- JobTracker
  - global Resource Manager - Cluster resource management
  - per application Application Master – doing job scheduling and monitoring, negotiating the resource containers from the Scheduler, tracking their status and monitoring for progress
- TaskTracker
  - new per-node slave Node Manager (NM) - responsible for launching the applications' containers, monitoring their resource usage (cpu, memory, disk, network) and reporting to the Resource Manager
  - (a per-application Container running on a NodeManager)
- YARN maintains compatibility with existing MapReduce applications and users

Hadoop MapReduce Classic

- JobTracker
  - Manages cluster resources and job scheduling
- TaskTracker
  - Per-node agent
  - Manage tasks



YARN – Architectural Overview



- Scalability - Clusters of 6,000-10,000 machines
  - Each machine with 16 cores, 48G/96G RAM, 24TB/36TB disks
  - 100,000+ concurrent tasks
  - 10,000 concurrent jobs

Comparing YARN with MapReduce

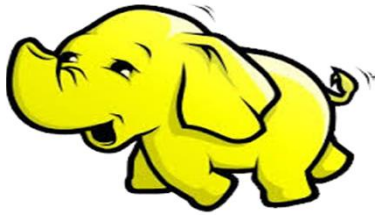
Criteria	**YARN**	**MapReduce**
Type of processing	Real-time, batch, interactive processing with multiple engines	Silo & batch processing with single engine
Cluster resource optimization	Excellent due to central resource management	Average due to fixed Map & Reduce slots
Suitable for	MapReduce & Non – MapReduce applications	Only MapReduce applications
Managing cluster resource	Done by YARN	Done by JobTracker

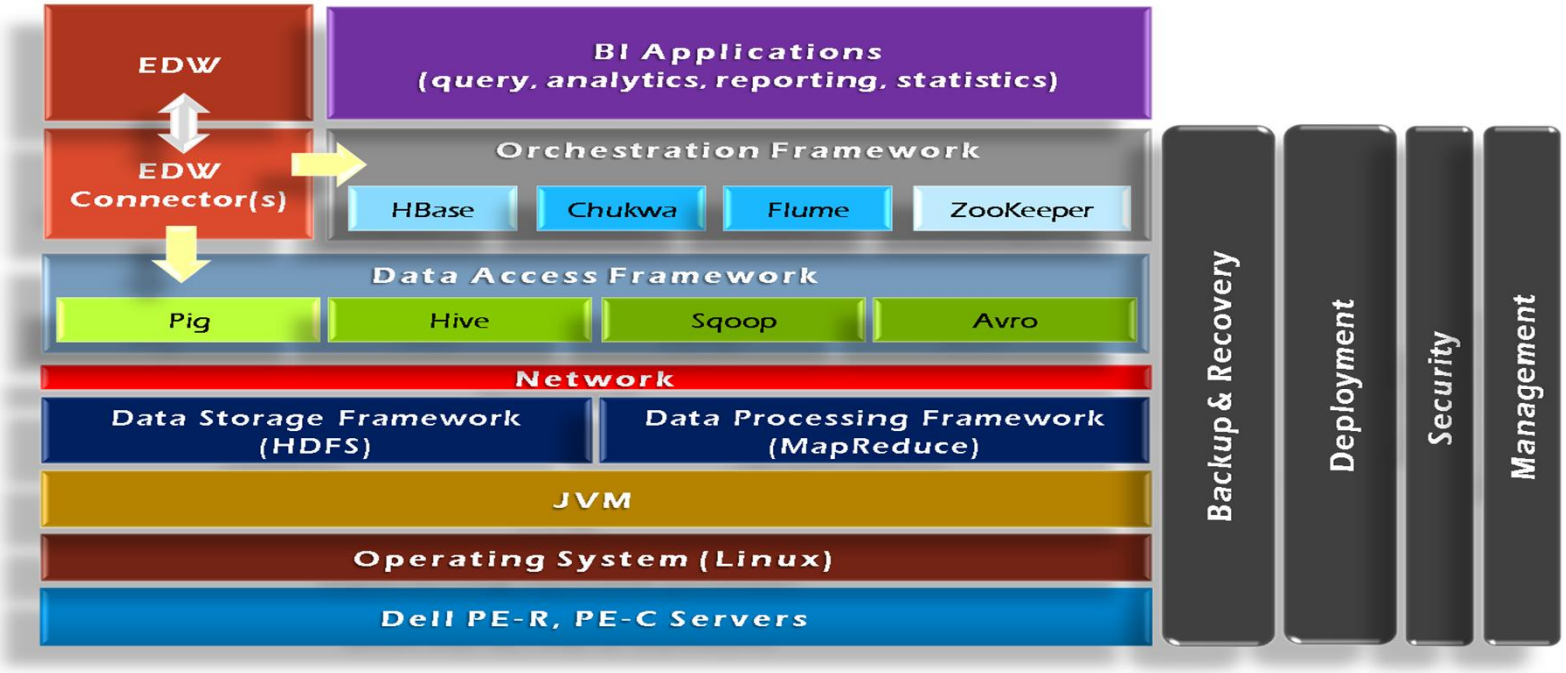
YARN advantage over MapReduce

- Support for programming paradigms other than MapReduce (Multi tenancy)
  - Tez – Generic framework to run a complex DAG
  - HBase on YARN (HOYA)
  - Compute engine (e.g., Machine Learning): Spark
  - Graph processing: Giraph
  - Real-time processing: Apache Storm
  - Enabled by allowing the use of paradigm-specific application master
  - *Run all on the same Hadoop cluster!*

Hadoop components

Hadoop Framework Tools





Hadoop Subprojects - Summary

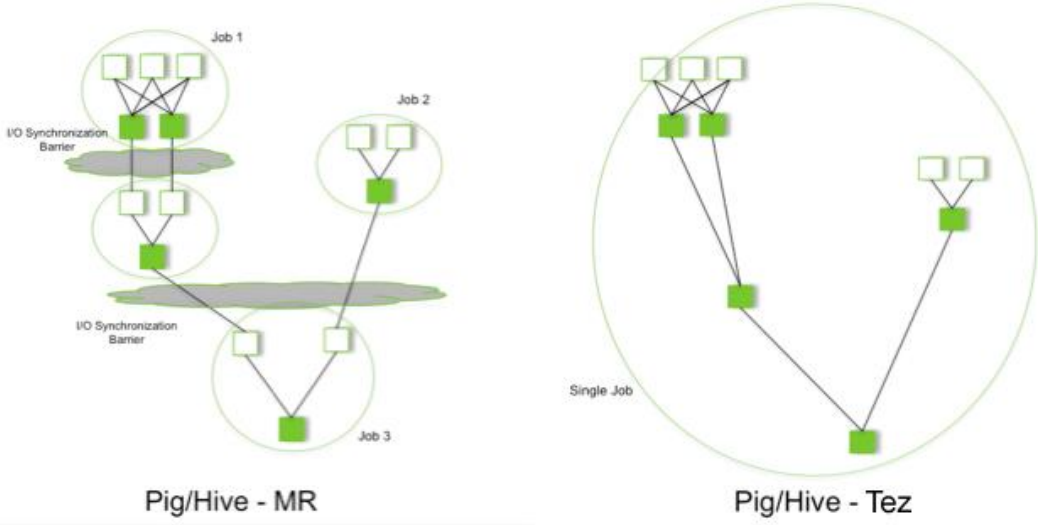
- Pig
  - High-level language for data analysis
- HBase
  - Table storage for semi-structured data
- Zookeeper
  - Coordinating distributed applications
- Hive
  - SQL-like Query language and Metastore
- Mahout
  - Machine learning

Tez on YARN

- Hindi for speed
- Provides a general-purpose, highly customizable framework that creates simplifies data-processing tasks across both small scale (low-latency) and large-scale (high throughput) workloads in Hadoop.
- Generalizes the MapReduce paradigm to a more powerful framework by providing the ability to execute a complex DAG
- Enables Apache Hive, Apache Pig and Cascading can meet requirements for human-interactive response times and extreme throughput at petabyte scale

Tez on YARN

- Original MapReduce requires disk I/O after each stage
- A series of MapReduce jobs following each other would result in lots of I/O
- Tez eliminates these intermediate steps, increasing the speed and lowering the resource usage



Pig/Hive - MR

Pig/Hive - Tez

Tez on YARN

- Performance gains over Mapreduce
  - Eliminates replicated write barrier between successive computations
  - Eliminates job launch overhead of workflow jobs
  - Eliminates extra stage of map reads in every workflow job
  - Eliminates queue and resource contention suffered by workflow jobs that are started after a predecessor job completes

HBase on YARN(HOYA)

- Be able to create on-demand HBase clusters easily -by and or in apps
  - With different versions of HBase potentially (for testing etc.)
- Be able to configure different HBase instances differently
  - For example, different configs for read/write workload instances
- Better isolation
  - Run arbitrary co-processors in user's private cluster
  - User will own the data that the hbase daemons create

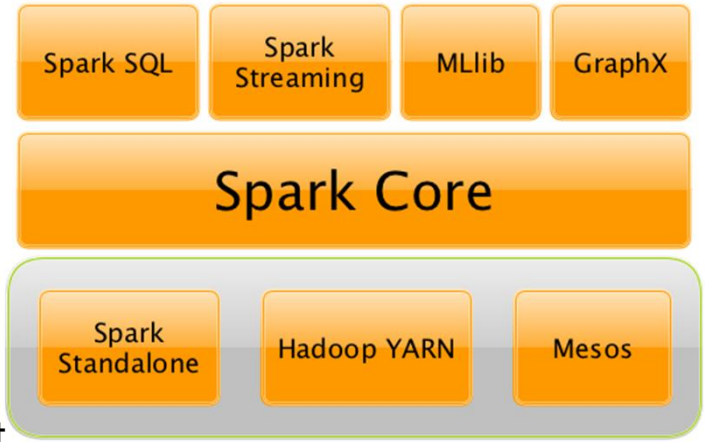
HBase on YARN(HOYA)

- MR jobs should find it simple to create (transient) HBase clusters
  - For Map-side joins where table data is all in HBase, for example
- Elasticity of clusters for analytic / batch workload processing
  - Stop / Suspend / Resume clusters as needed
  - Expand / shrink clusters as needed
- Be able to utilize cluster resources better
  - Run MR jobs while maintaining HBase's low latency SLAs

SPARK

Apache Spark

- An Apache Foundation open source project. Not a product.
- Standalone generic BigData computational framework
  - Both batch and streaming mode
- An in-memory compute engine that works with data
  - Not a data store
- Enables highly iterative analysis on large volumes of data at scale
- Unified environment for data scientists, developers and data engineers
- Radically simplifies process of developing intelligent apps fueled by data
- Can be combined with Hadoop
  - But can work without Hadoop, too: e.g., Kubernetes




Relationship with Apache Mesos
The two projects go back a long time!
2008: Mesos started as UC Berkeley research project
#MesosCon
NORTH AMERICA

Key reasons for Spark

High Performance

- In-memory architecture greatly reduces disk I/O
- Anywhere from **20-100x faster** for common tasks

Productive

- **Concise and expressive syntax**, especially compared to prior approaches (up to 5x less code)
- **Single programming model** across a range of use cases and steps in data lifecycle
- **Integrated with common programming languages** – Java, Python, Scala
- **New tools** continually reduce skill barrier for access (e.g. SQL for analysts)

Leverages existing investments

- Works well within **existing Hadoop ecosystem**

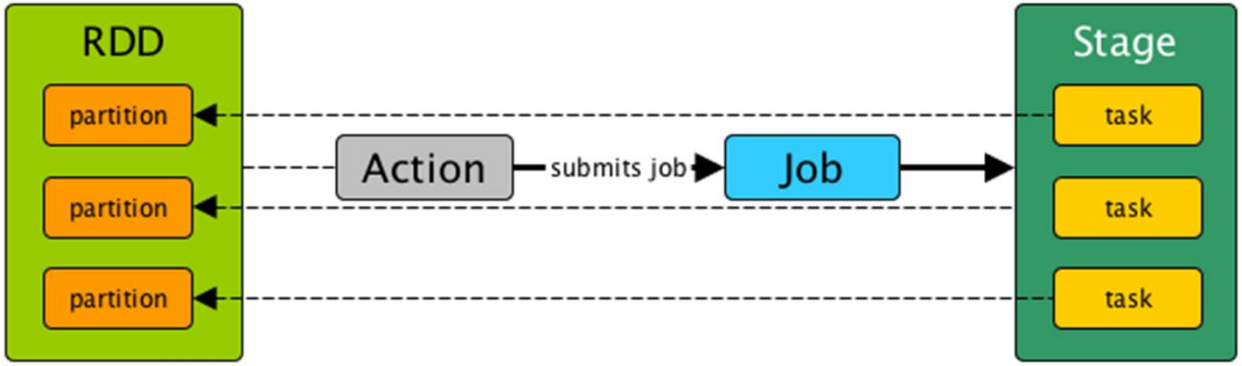
Improves with age

- **Large and growing community** of contributors continuously improve full analytics stack and extend capabilities

S. Pandey: Spark as a Service, IBM Cloud Data Services

Spark components

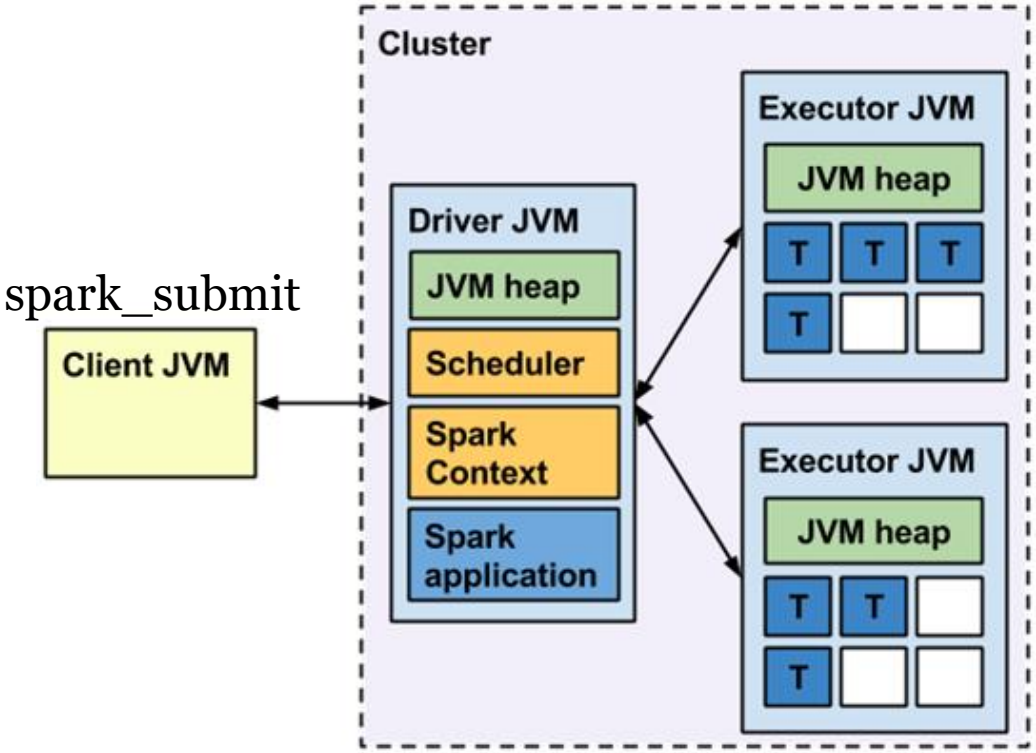
- **Resilient Distributed Dataset (RDD)**
  - The primary data abstraction and the core of Spark
  - Resilient and distributed collection of records spread over many partitions
  - Shuffling: redistributing data across partitions
- **Stage**
  - Physical unit of execution



Spark terminology

Driver	*Process that contains the SparkContext*
Executor	*Process that executes one or more Spark tasks*
Master	*Process that manages applications across the cluster*
Worker	*Process that manages executors on a particular node*

Spark / cluster mode deployment

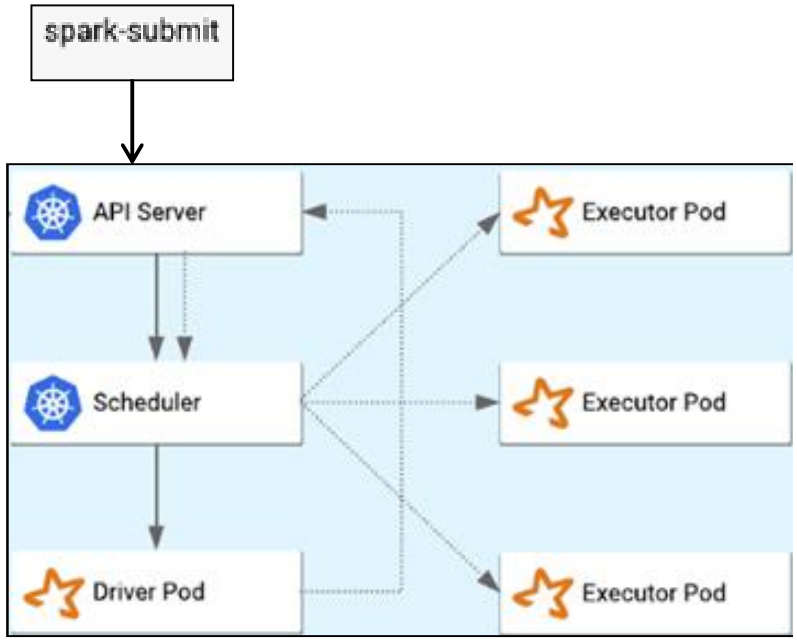


Static Spark on Kubernetes

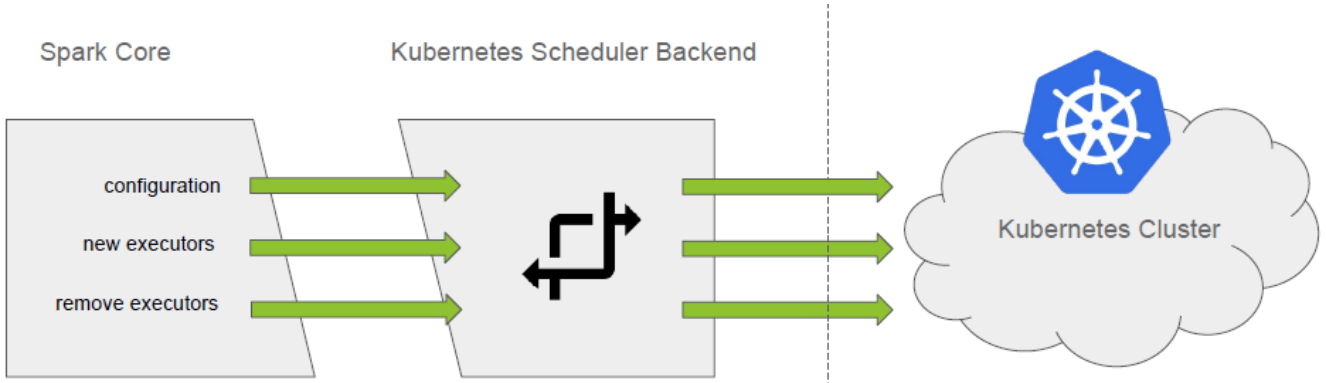
- A Single pod is created for Spark Master
- For all workers, there will pod for each worker
- All the pods runs custom built spark image
- These pods are connected using kubernetes networking abstractions
- This creates a static spark cluster on kubernetes
- As the spark is not aware it's not running on kubernetes , it doesn't recognise the limits put on kubernetes pods
- For ex: In kubernetes we can define pod to have 1 GB RAM, but we may end up configure spark worker to have 10 GB memory
- This mismatch in resource definition makes it tedious in keeping both in sync
- The same applies for CPU and Disk bounds also

Kubernetes is the new YARN (for Spark)

- When spark is deployed on YARN, spark treats YARN as a container management system
- Spark requests the containers from YARN with defined resources
- Once it acquires the containers, it builds a RPC based communication between containers to run driver and executors
- Spark can scale automatically by releasing and aquiring containers
- From Spark 2.3, spark supports kubernetes as new cluster backend
- It adds to existing list of YARN, Mesos and standalone backend
- This is a native integration, where no need of static cluster is need to built before hand
- Works very similar to how spark works yarn



Spark on K8S

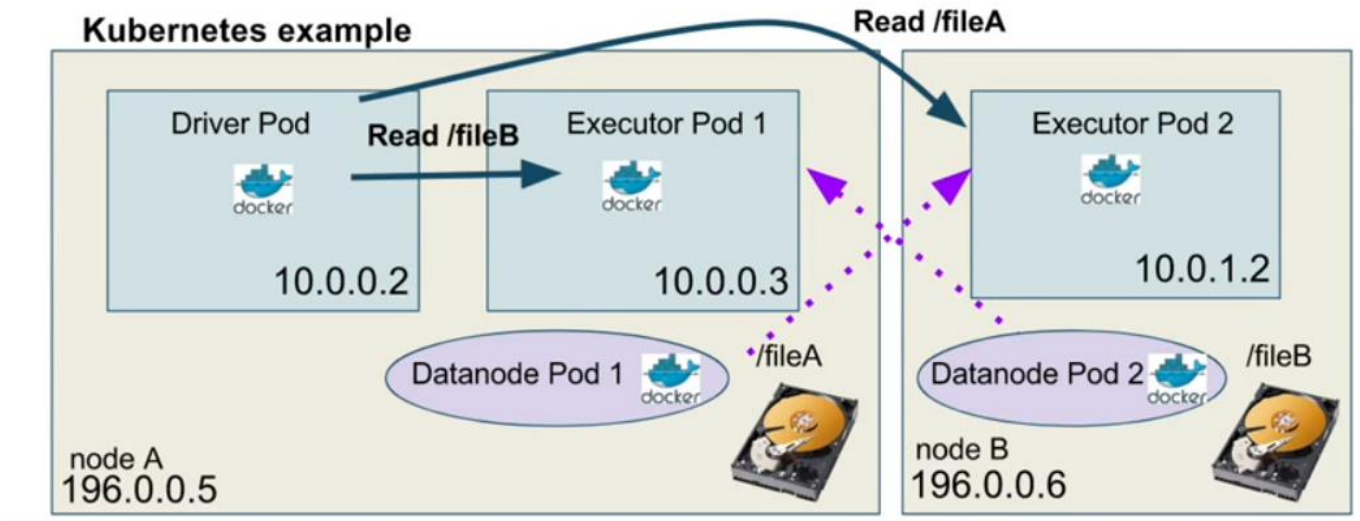


- Resource Requests
- Authnz
- Communication with K8s

- Runs Spark Drivers/Executors
- Runs Shuffle Service
- Runs Additional Components for Spark jobs

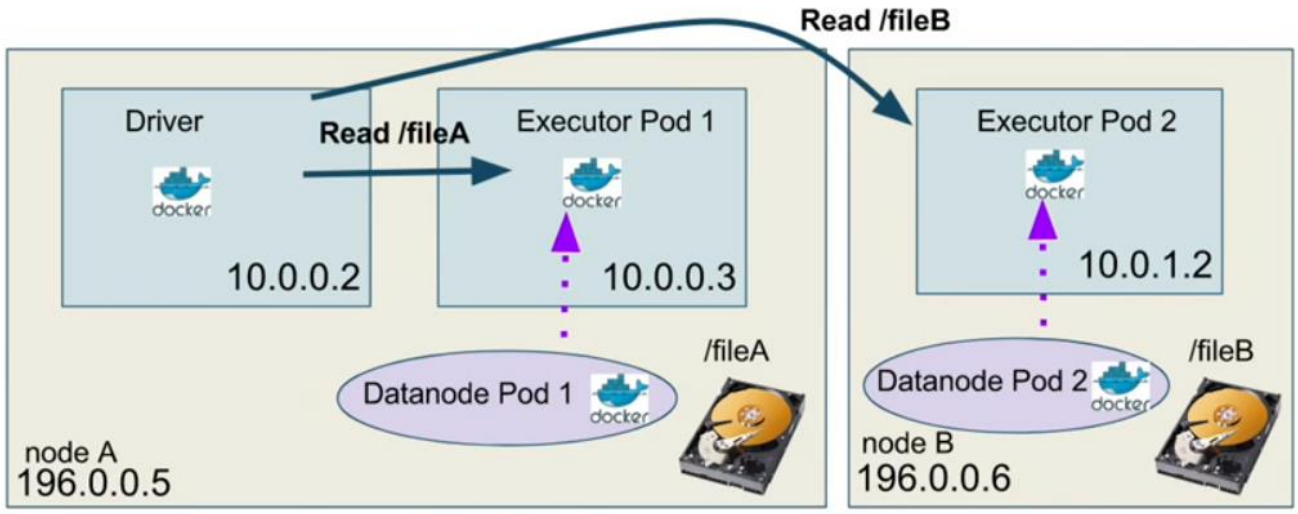
Spark on K8s + HDFS

- No YARN, no data locality?

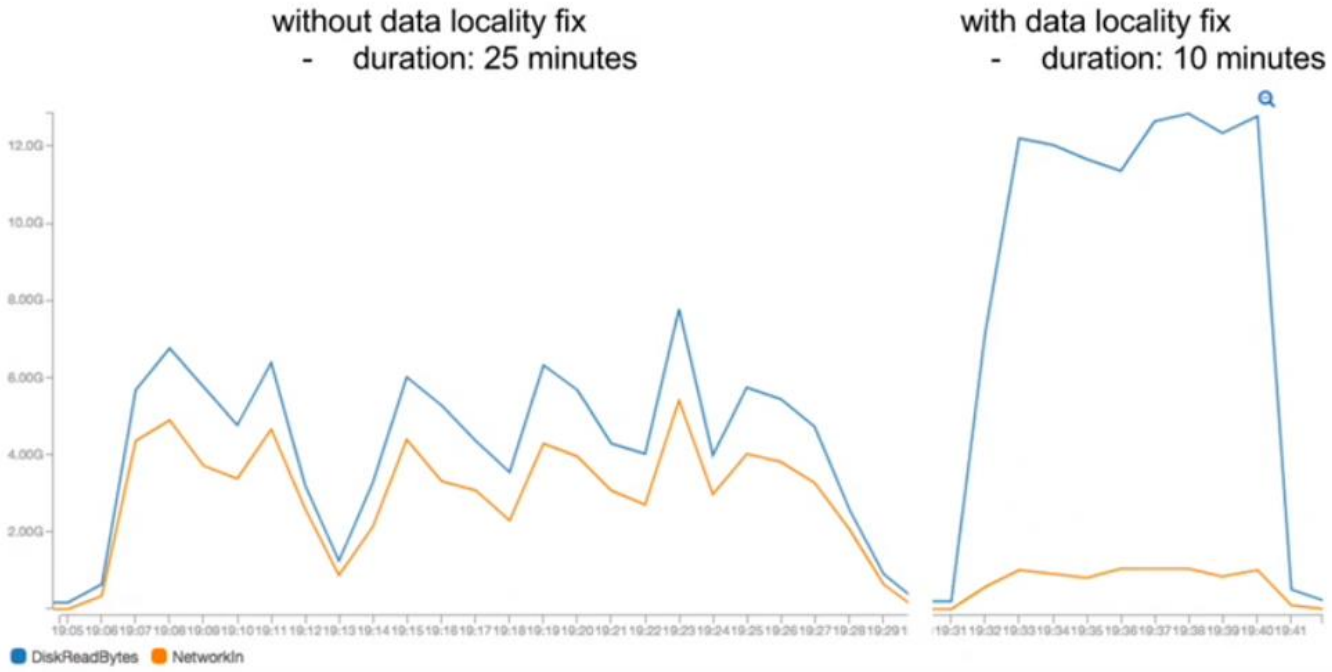


Spark on K8s + HDFS

- K8s master provides an API to match executor and datanode host IDs (IP, label)

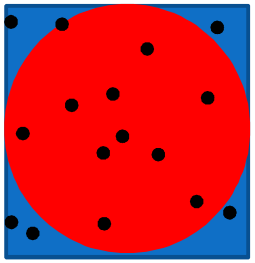


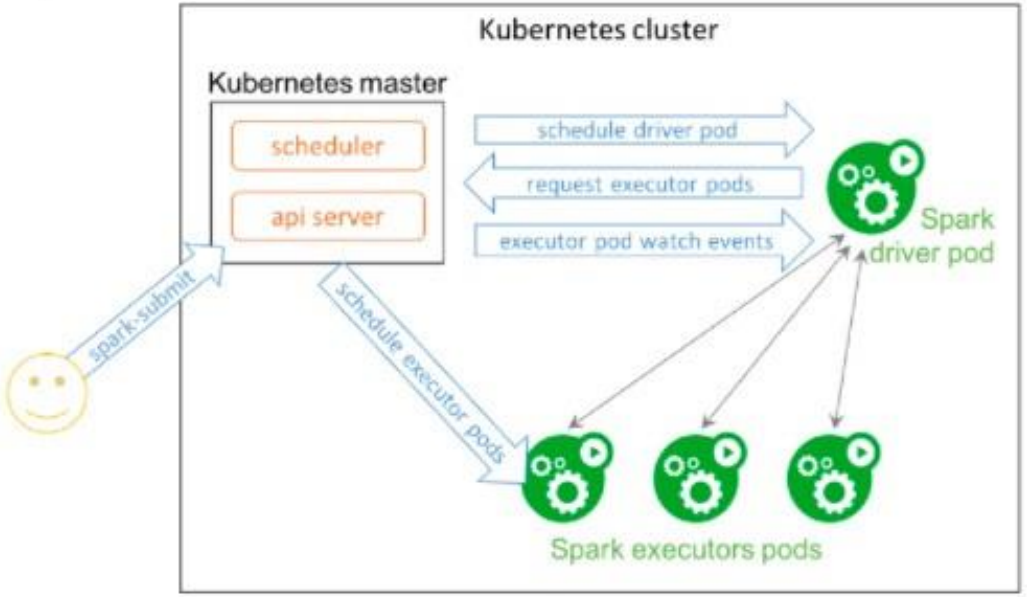
The effect of data locality



SparkPI example over Kubernetes

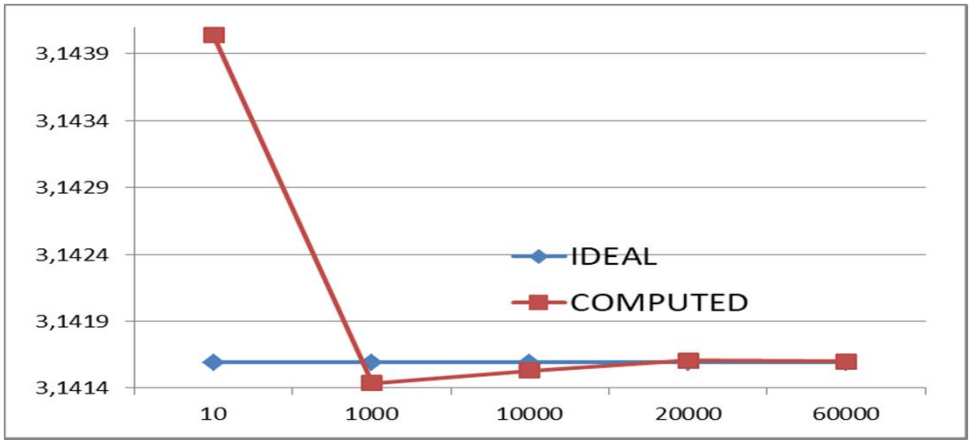
- Kubernetes Custom Controller is an extension to kubernetes API to defined and create custom resources in Kubernetes
- Spark uses customer controller to create spark driver which interns responsible for creating worker pods



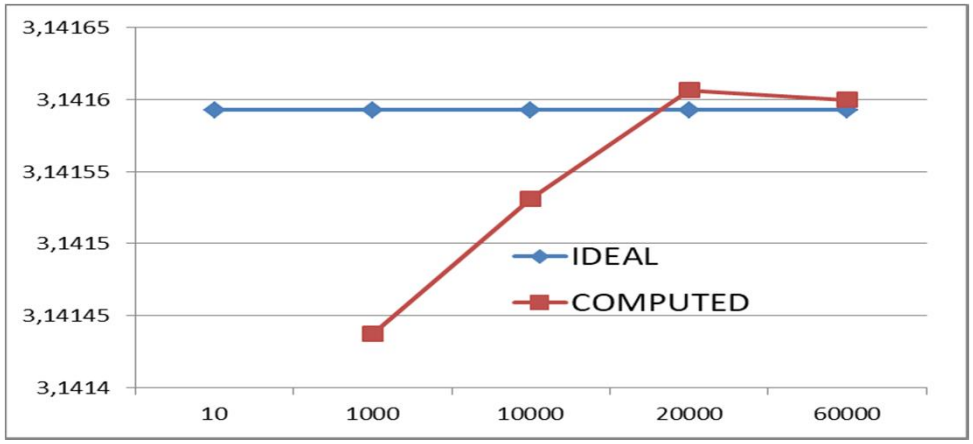


Apache Spark running natively in a Kubernetes cluster

Pi számítása BigData klaszterben

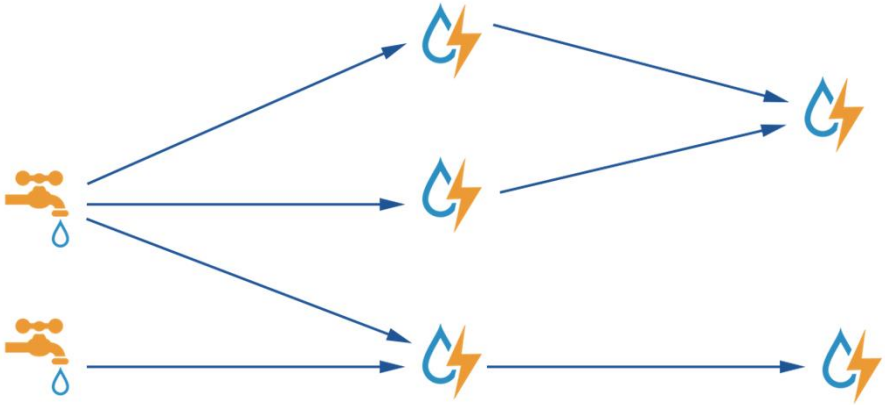


Pi számítása BigData klaszterben



RealTime data processing: Batch vs. Streaming

- Apache Storm



- Spark Streaming

- (message queuing: Kafka)
- (RPC: gRPC)