Spark:

Cluster Computing with Working Sets

Medical Compute with ChRIS on the MOC
PowerPC & x86_64 GPU Usage & Benchmarking

01 Introduction To Spark

02 Spark Programming Model

03 Spark System Architecture

04 Spark Demo

Introduction to Spark

What is Spark

Mapreduce' success and limitations

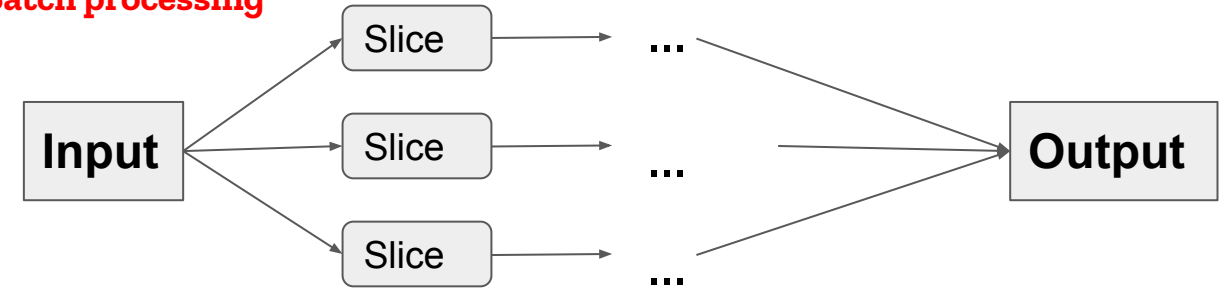
Spark's Charm

What is Spark?

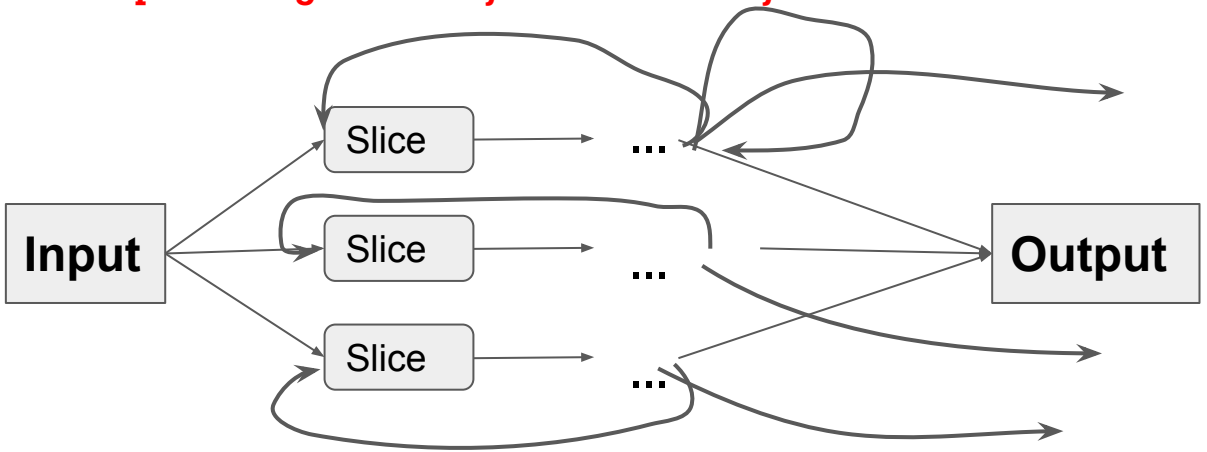
A: Cluster Computing Framework

- Mapreduce' success and limitations
- Spark's Charm...

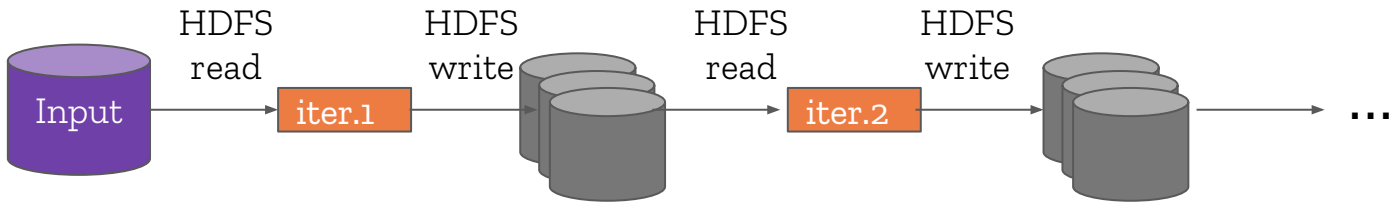
Batch processing



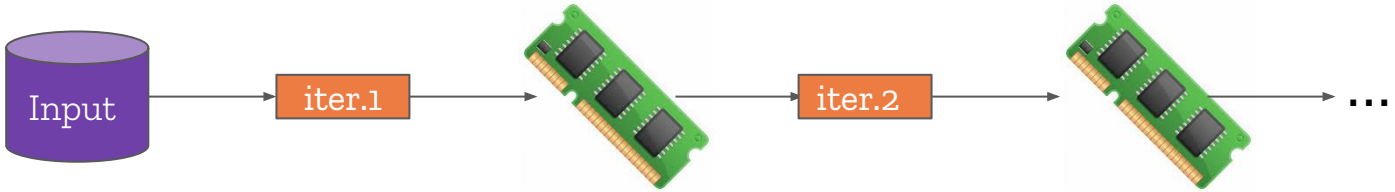
Real-time processing /iterative jobs/ interactive jobs



MapReduce

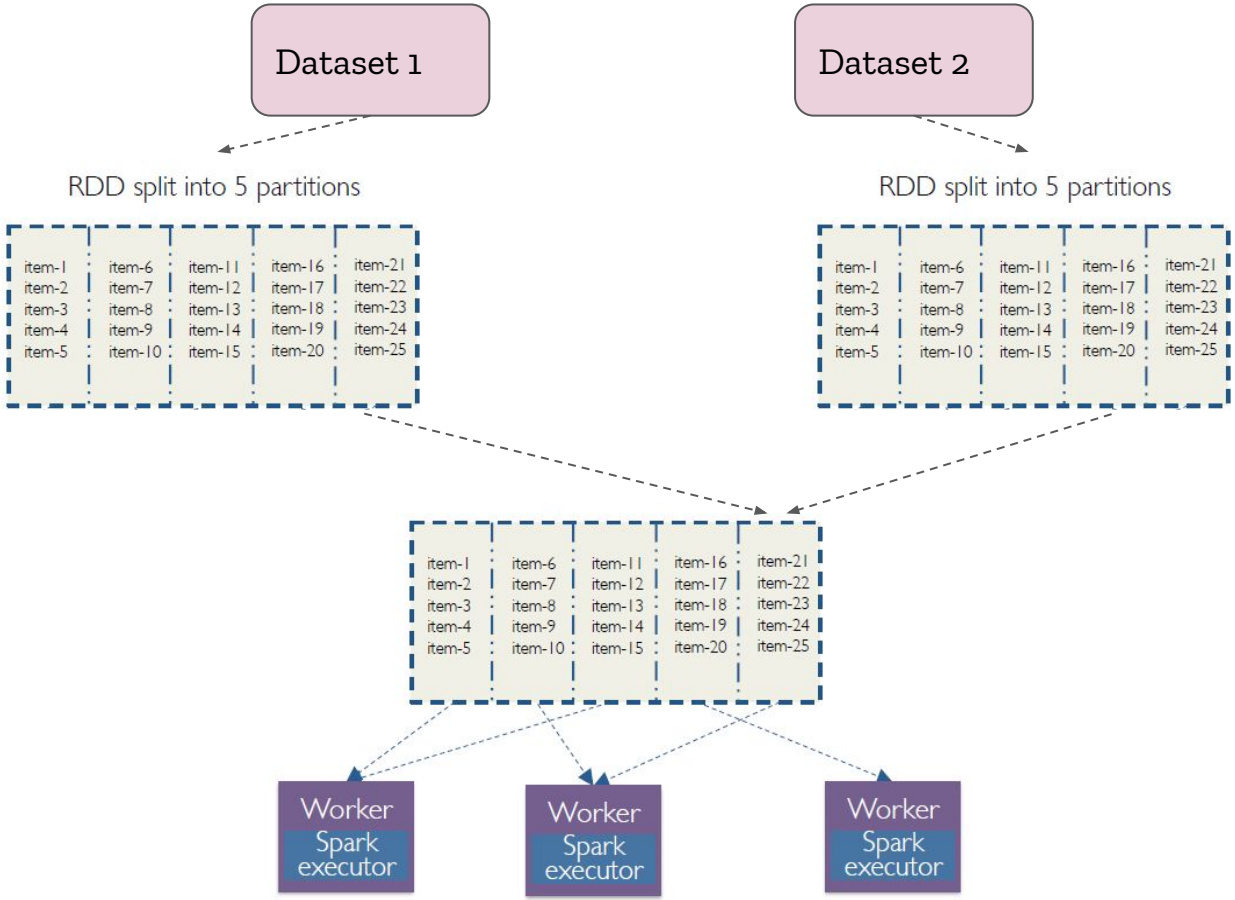


Spark

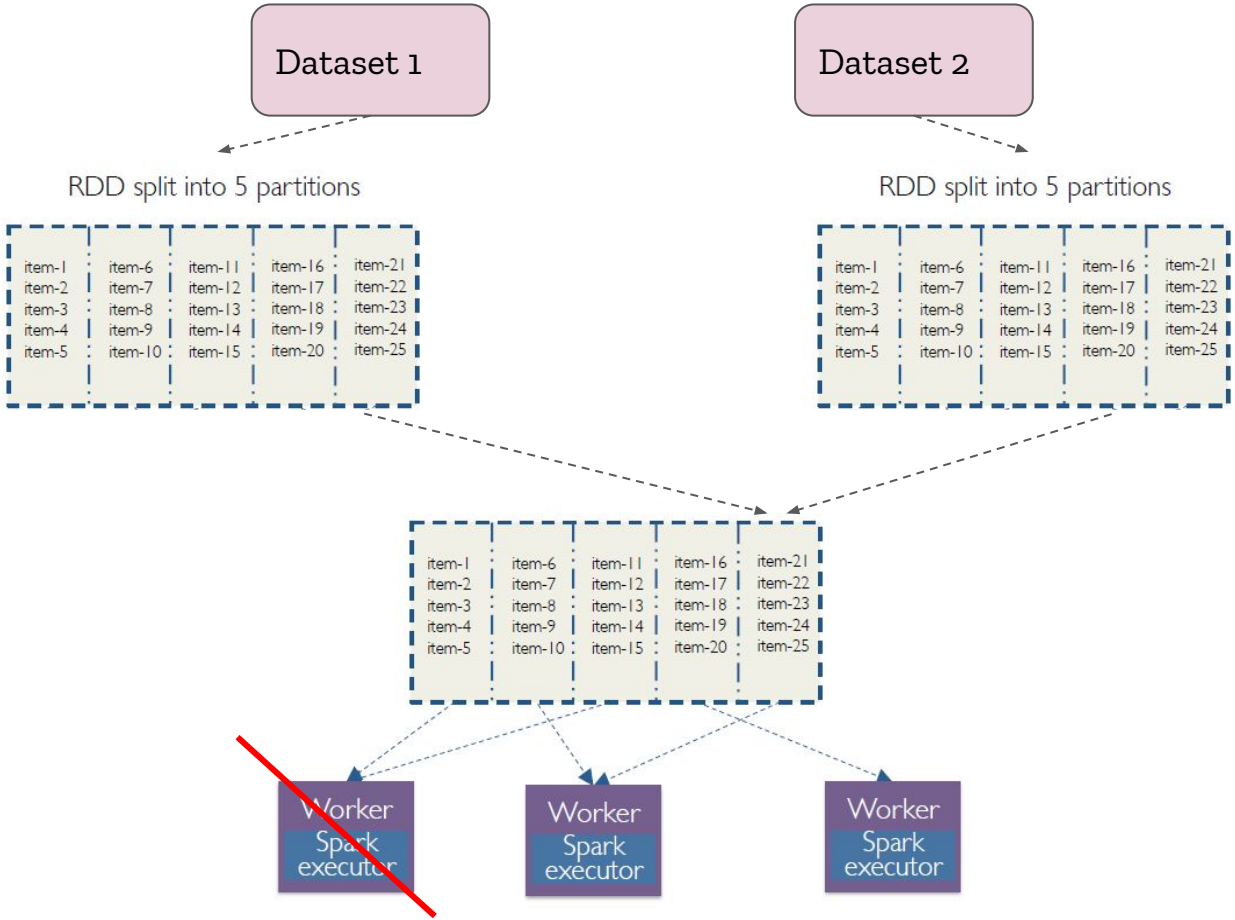


I/O cost of disk is way much higher than memory

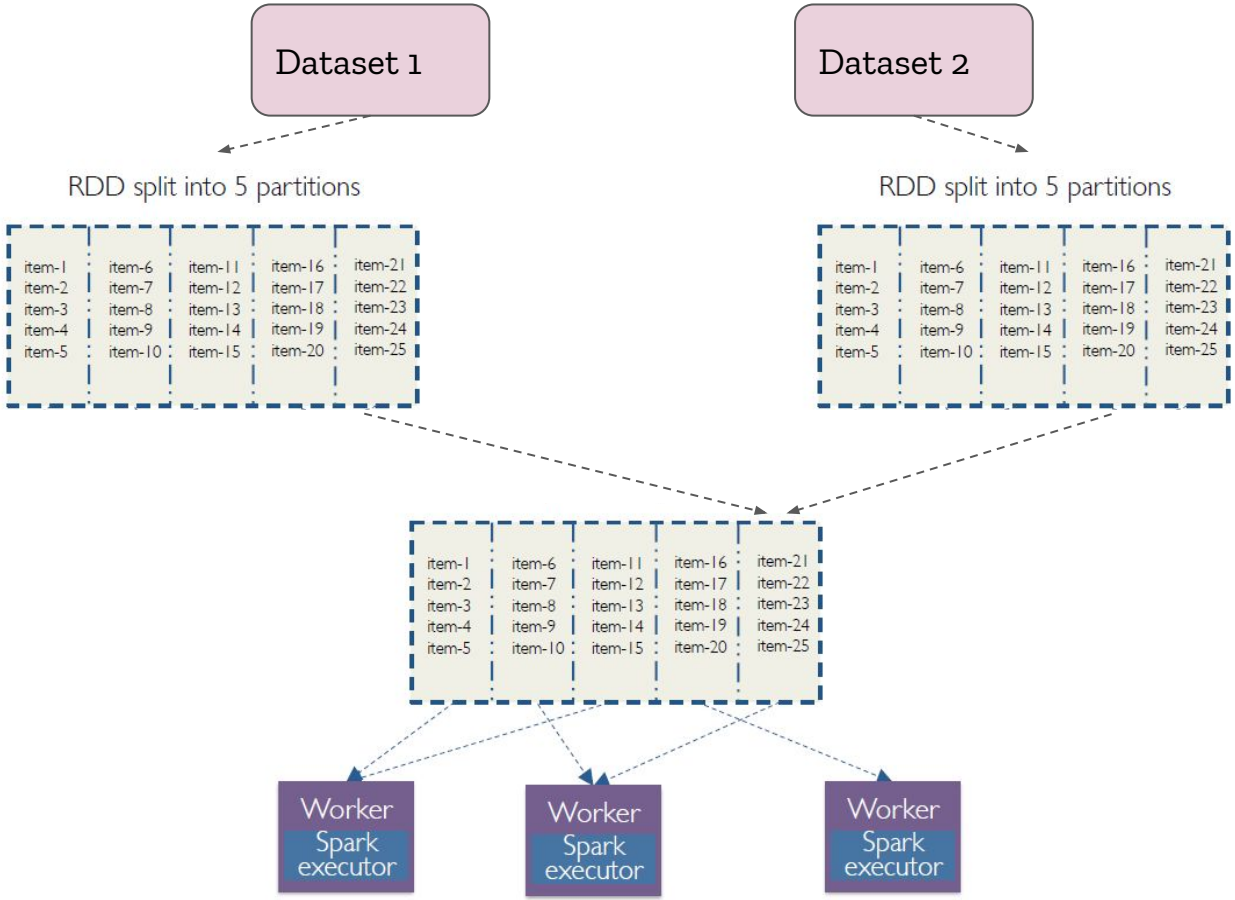
Fundamental Core of Spark



Fundamental Core of Spark

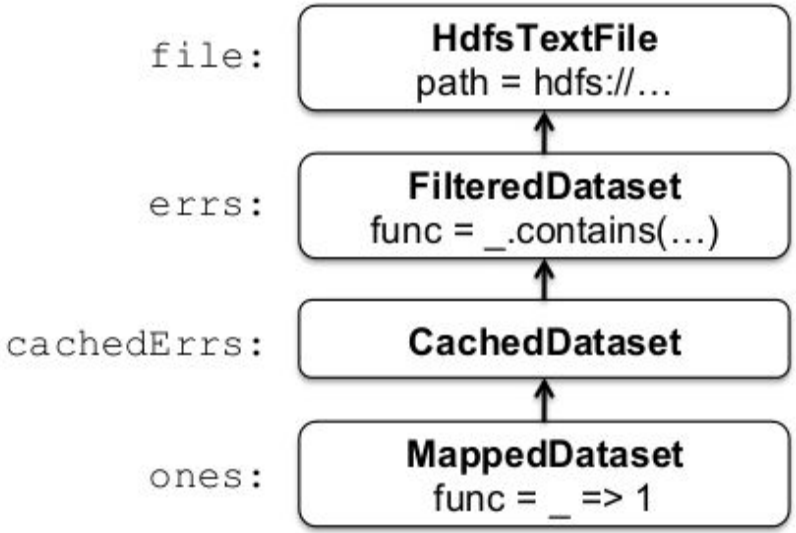


Fundamental Core of Spark

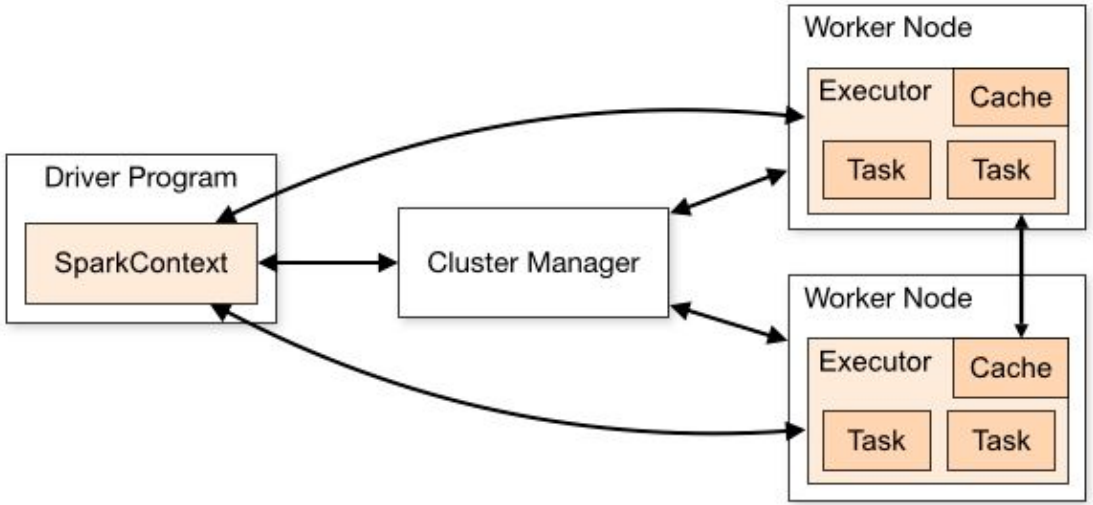


Spark RDD Implementation

- Hierarchy where children point to parents
- **`HdfsTextFile`**
  - Original dataset
- **`FilteredDataset`**
  - Dataset with only errors
- **`CachedDataset`**
  - Locally cached copy of transformed partition
- **`MappedDataset`**
  - Iterator applies a map function to elements of the parent



Spark Framework Architecture



Spark's Programming Model

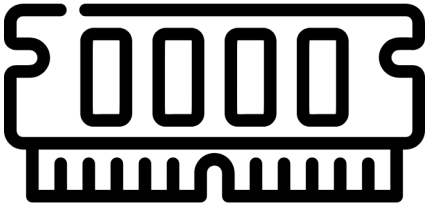
Contents

Resilient Distributed Dataset (RDD)

a fault-tolerant collection of elements
that can be operated on in parallel

What is Resilient Distributed Datasets (RDD)

a **read-only** collection of objects



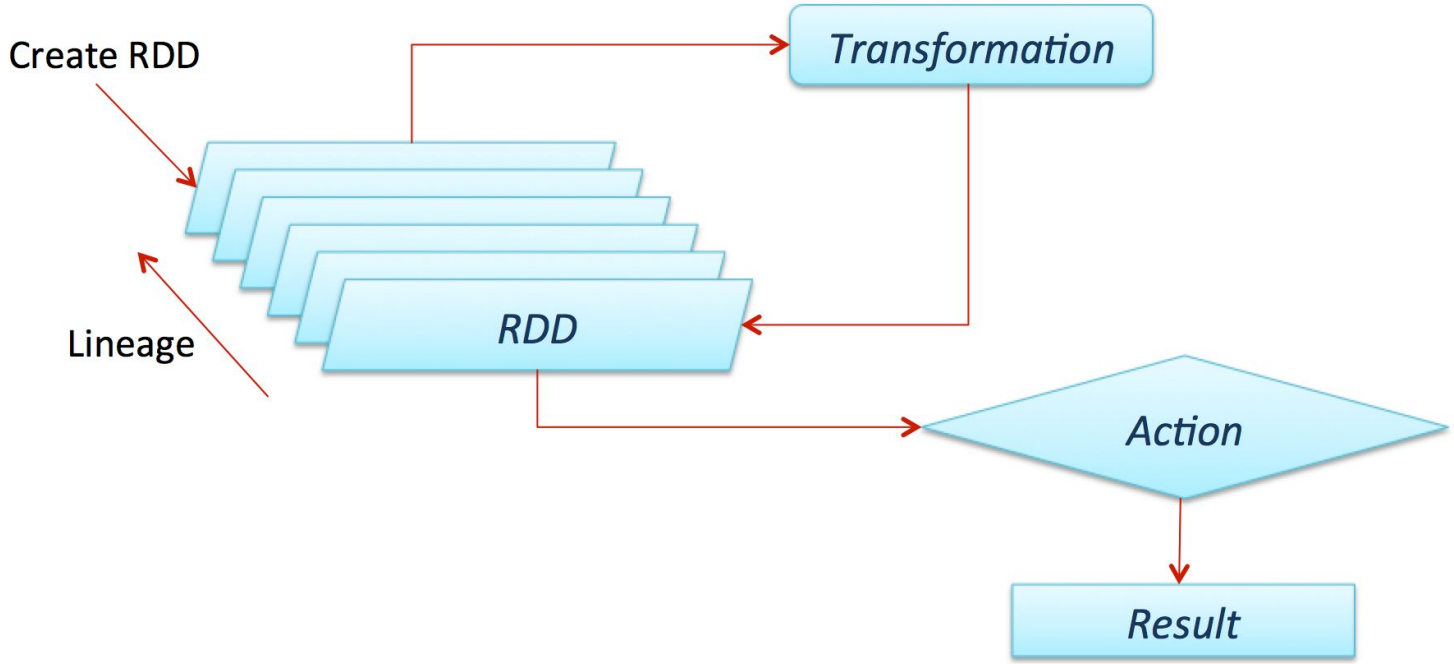
Allows user load a dataset into **memory**

fault-tolerant collection

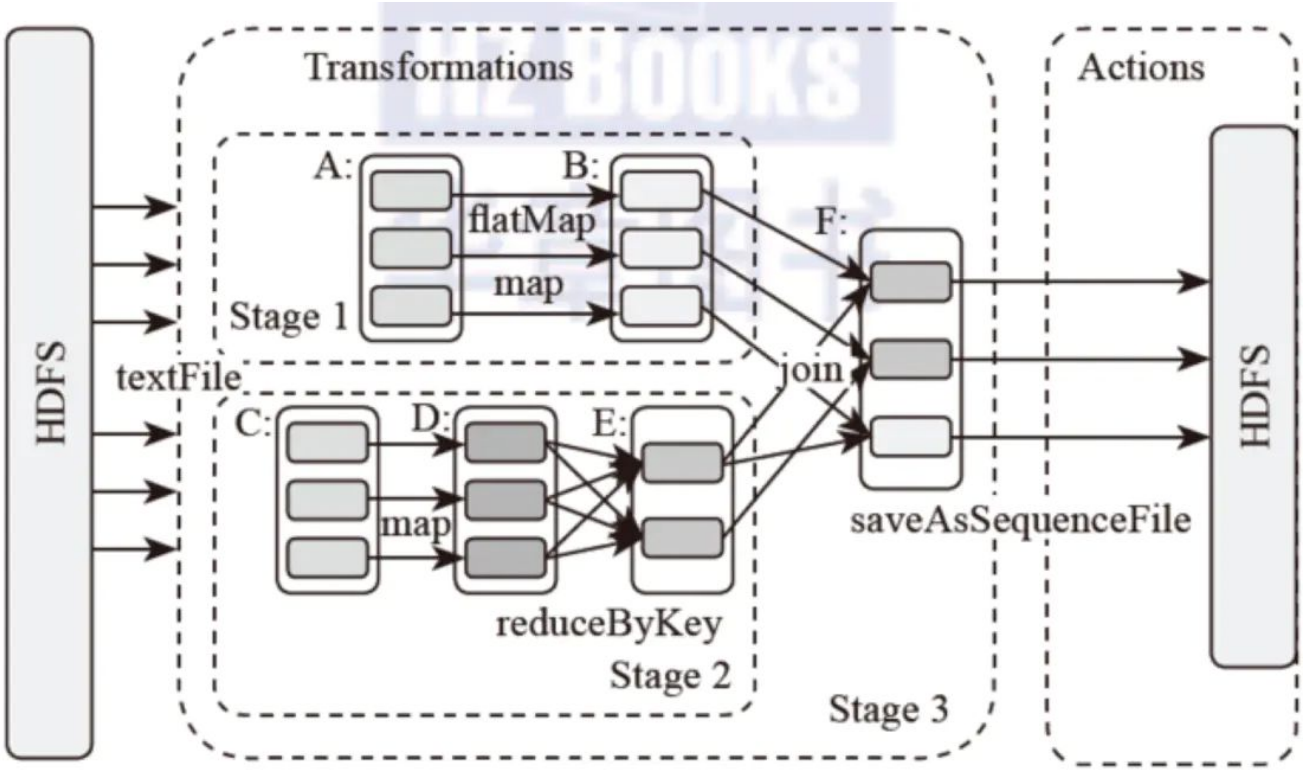
- **Re-computable**
  - Lazy Computation
    - Spark doesn't compute task on `transformation operators`, only compute it on `Action operators`
    - a handle contains information to compute RDDs starting from the first RDD.
  - Ephemeral Storage
    - If there is not enough memory in cache all partitions of a dataset, Spark may drop some and recompute them when they are used.

A ways to Create RDD

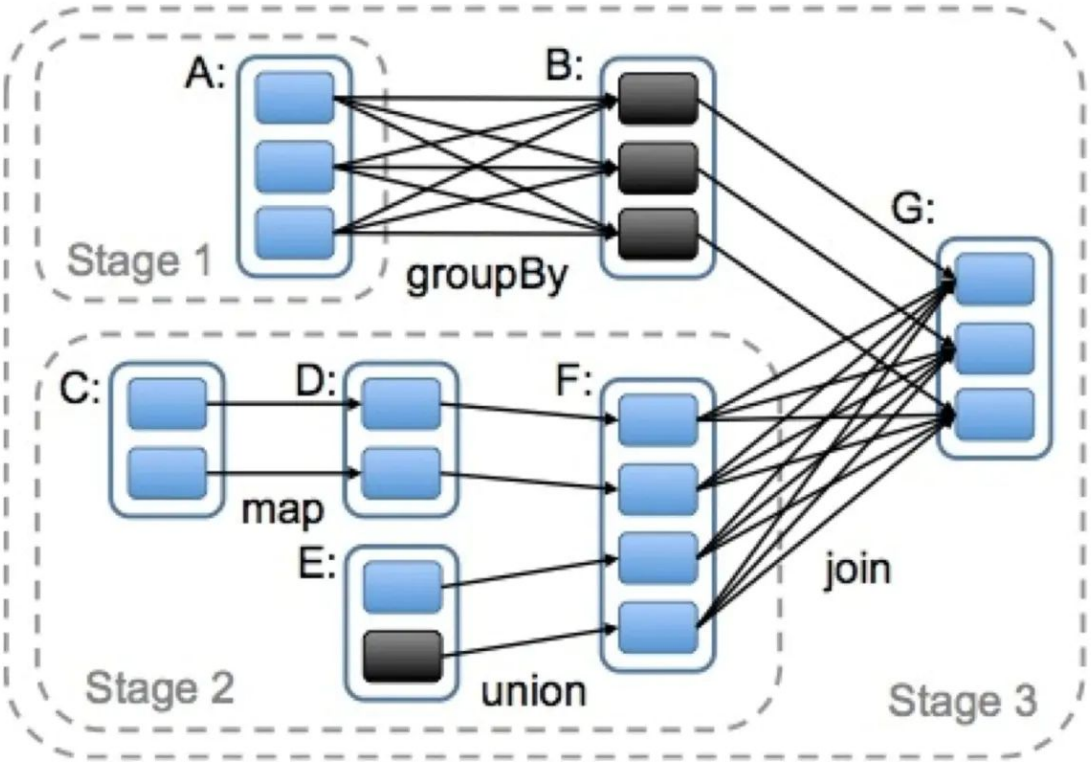
1. Parallelizing already existing collection in driver program
2. Referencing a dataset in an external storage system (e.g. HDFS, Hbase, shared file system).
3. Creating RDD from already existing RDDs.


Create RDD
Transformation
RDD
Lineage
Action
Result

How RDD works



Stages, Narrow Dependency & Wide Dependency



Operations in Spark

A ways to Create RDD

1. Parallelizing already existing collection in driver program
2. Referencing a dataset in an external storage system (e.g. HDFS, Hbase, shared file system).
3. Creating RDD from already existing RDDs.

A ways to Create RDD

1. Parallelizing already existing collection in driver program
2. Referencing a dataset in an external storage system (e.g. HDFS, Hbase, shared file system).
3. **Transformation**

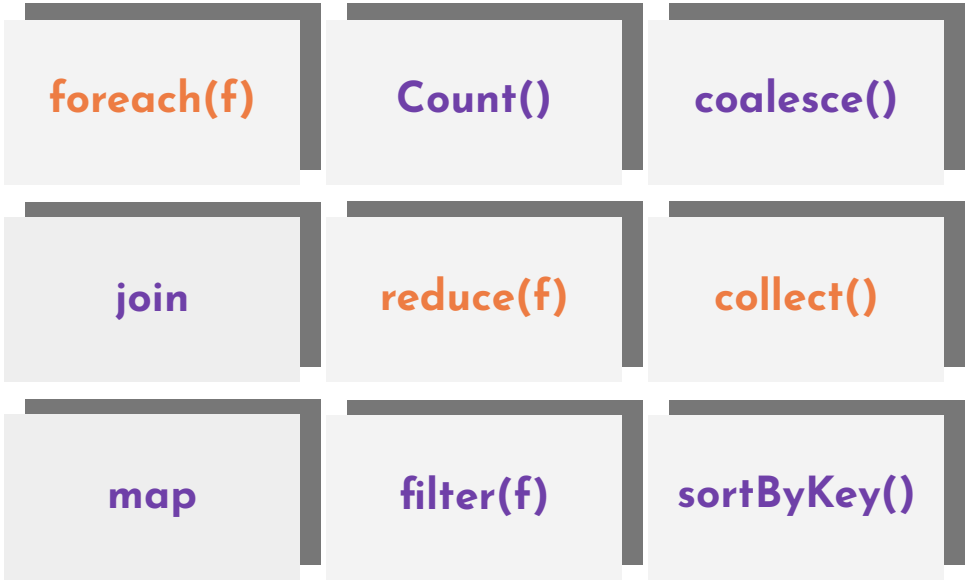
Types of Spark RDD Operations

Transformation

A function that produces new RDD from the existing RDDs

Action

A function to work with the actual dataset



Operations in Spark

Parallel Operations

Reduce

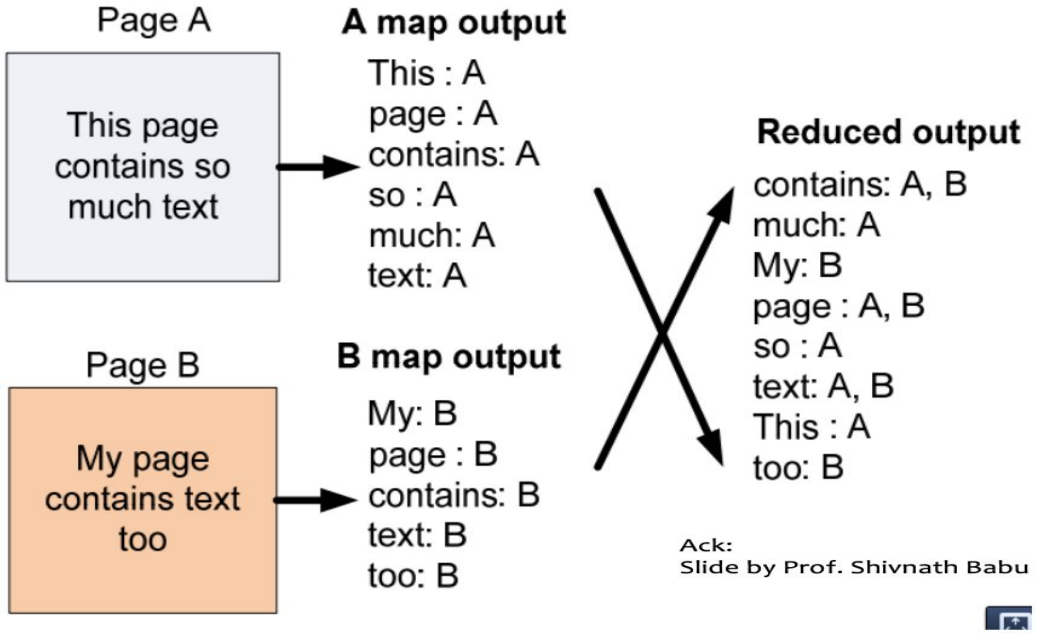
Combines dataset elements using an associative function to produce result

Collect

Sends all element of dataset to the driver program

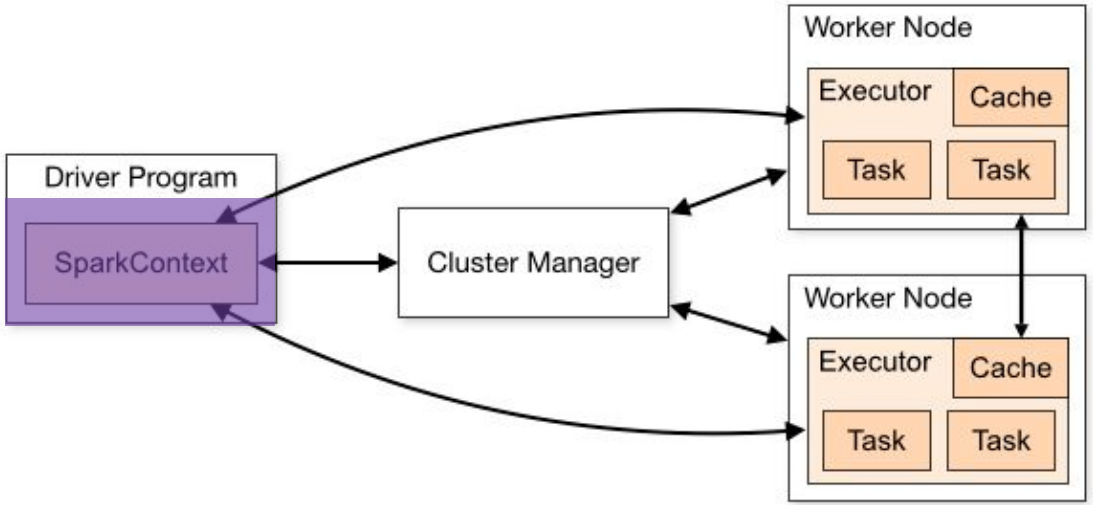
Foreach

Passes each element through user provided function


Page A
This page contains so much text
A map output
This : A
page : A
contains: A
so : A
much: A
text: A
Page B
My page contains text too
B map output
My: B
page : B
contains: B
text: B
too: B
Reduced output
contains: A, B
much: A
My: B
page : A, B
so : A
text: A, B
This : A
too: B
Ack:
Slide by Prof. Shivnath Babu

Reduce

Spark Framework's Interaction with Clusters



Example Code: Reduce(f)

```
from pyspark import SparkContext
from operator import add
sc = SparkContext("local", "Reduce app")
nums = sc.parallelize([1, 2, 3, 4, 5])
adding = nums.reduce(add)
print "Adding all the elements -> %i" % (adding)
```

Output:
Adding all the elements -> 15

Example Code: Reduce(f)

```
from pyspark import SparkContext
from operator import add
sc = SparkContext("local", "Reduce app")
nums = sc.parallelize([1, 2, 3, 4, 5])
adding = nums.reduce(add)
print "Adding all the elements -> %i" % (adding)
```

Output:
Adding all the elements -> 15

Example Code: Reduce(f)

```
from pyspark import SparkContext
from operator import add
sc = SparkContext("local", "Reduce app")
nums = sc.parallelize([1, 2, 3, 4, 5])
adding = nums.reduce(add)
print "Adding all the elements -> %i" % (adding)
```

Output:
Adding all the elements -> 15

Example Code: Reduce(f)

```
from pyspark import SparkContext
from operator import add
sc = SparkContext("local", "Reduce app")
nums = sc.parallelize([1, 2, 3, 4, 5])
adding = nums.reduce(add)
print "Adding all the elements -> %i" % (adding)
```

Output:
Adding all the elements -> 15

A ways to Create RDD

1. **Parallelizing already existing collection in driver program**
2. Referencing a dataset in an external storage system (e.g. HDFS, Hbase, shared file system).
3. Creating RDD from already existing RDDs.

Example Code: Reduce(f)

```
from pyspark import SparkContext
from operator import add
sc = SparkContext("local", "Reduce app")
nums = sc.parallelize([1, 2, 3, 4, 5])
adding = nums.reduce(add)
print "Adding all the elements -> %i" % (adding)
```

Output:
Adding all the elements -> 15

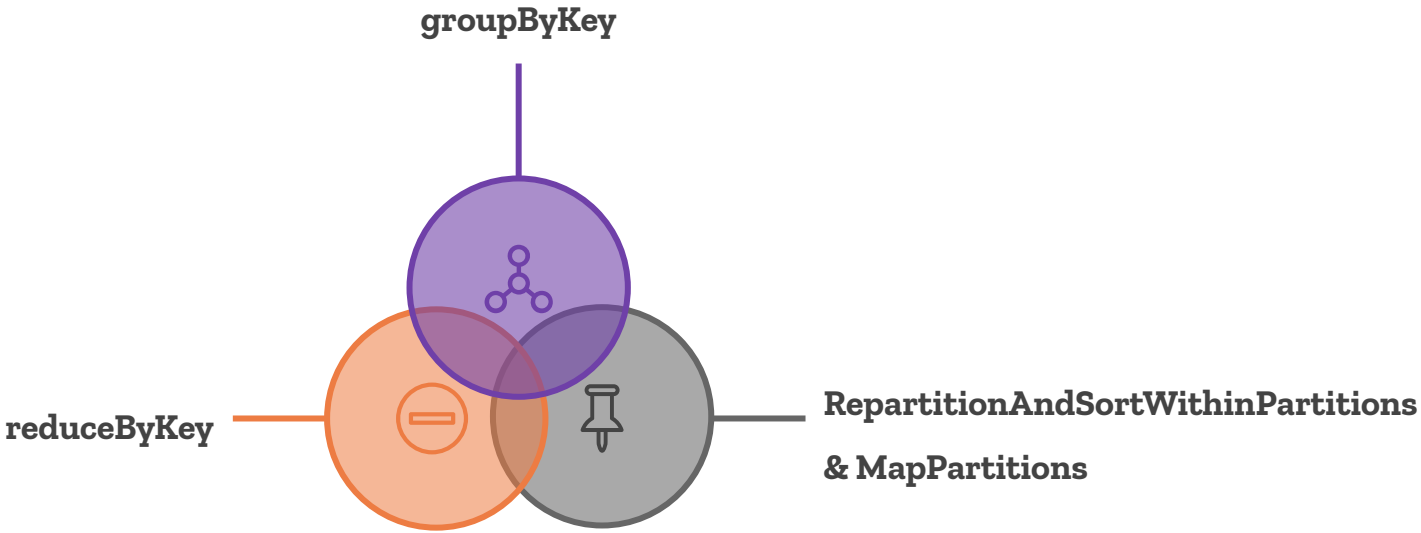
Example Code: Reduce(f)

```
from pyspark import SparkContext
from operator import add
sc = SparkContext("local", "Reduce app")
nums = sc.parallelize([1, 2, 3, 4, 5])
adding = nums.reduce(add)
print "Adding all the elements -> %i" % (adding)
```

Output:
Adding all the elements -> 15

Spark doesn't support a
Grouped Reduce Operation
as in MapReduce
In 2010

Grouped Reductions



Example Code: Collect

```
from pyspark import SparkContext
sc = SparkContext("local", "Collect app")
words = sc.parallelize (
 ["scala",  "Java", "hadoop", "Spark",  "Akka",
 "spark vs hadoop", "Pyspark",  "pyspark and spark"])
coll = words.collect()
print "Elements in RDD -> %s" % (coll)
```

Output:
Elements in RDD -> [
'Scala', 'Java', 'hadoop', 'Spark', 'Akka', 'spark vs hadoop', 'Pyspark', 'pyspark and spark']

Example Code: Foreach

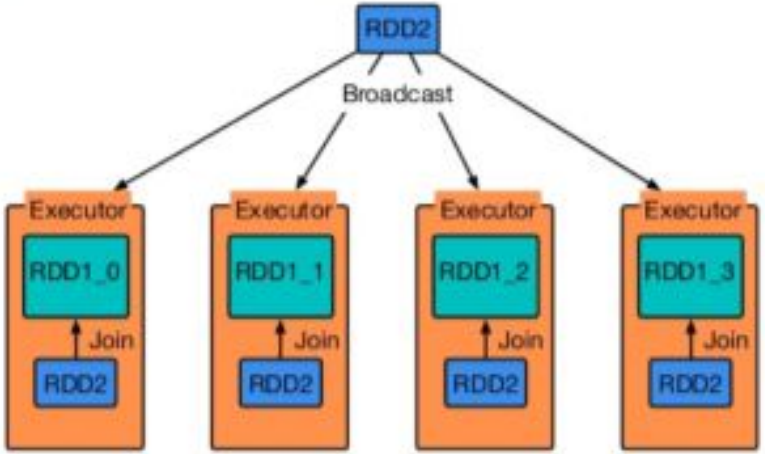
```
from pyspark import SparkContext
sc = SparkContext("local", "ForEach app")
words = sc.parallelize (
 ["scala", "java", "hadoop", "spark", "akka",
 "spark vs hadoop",  "pyspark", "pyspark and spark"]
)
def f(x): print(x)
fore = words.foreach(f)
```

Output:
Scala java Hadoop spark akka spark vs hadoop pyspark pyspark and spark

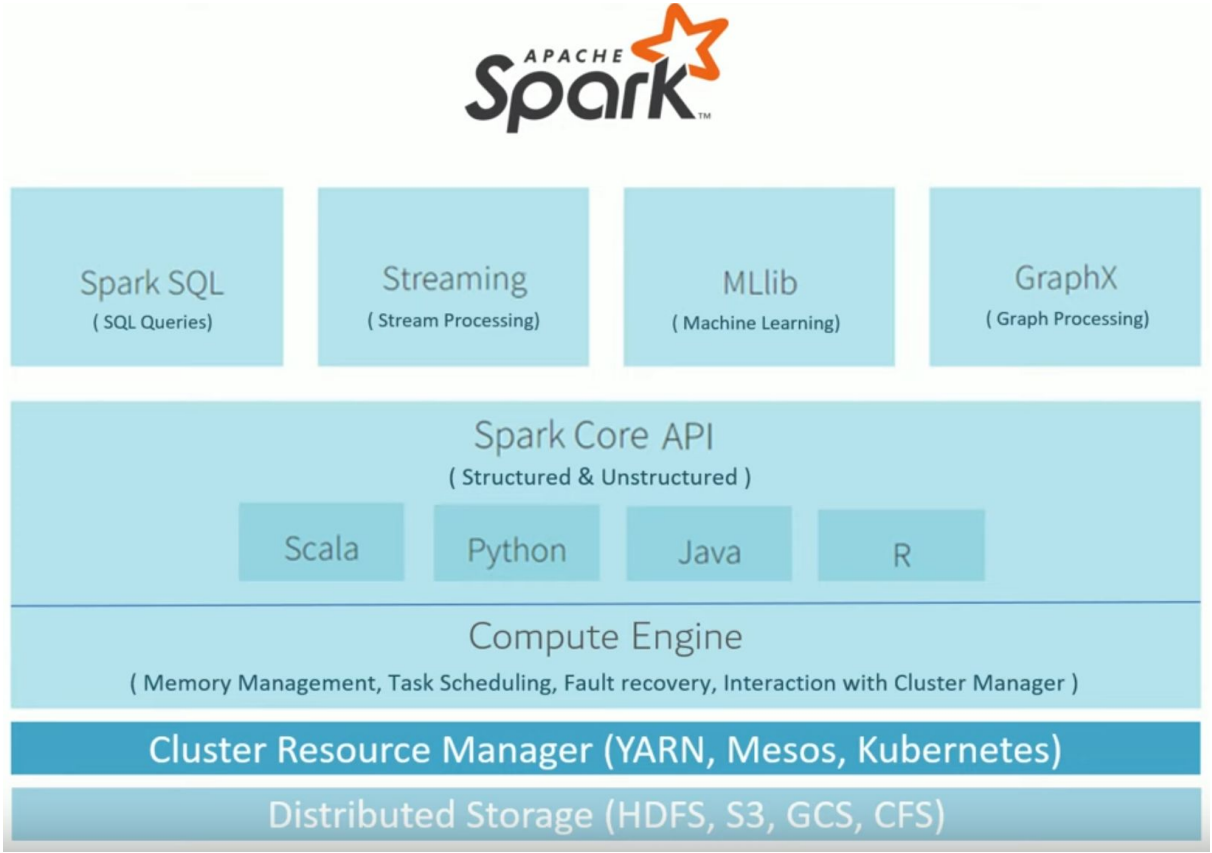
Shared Variables

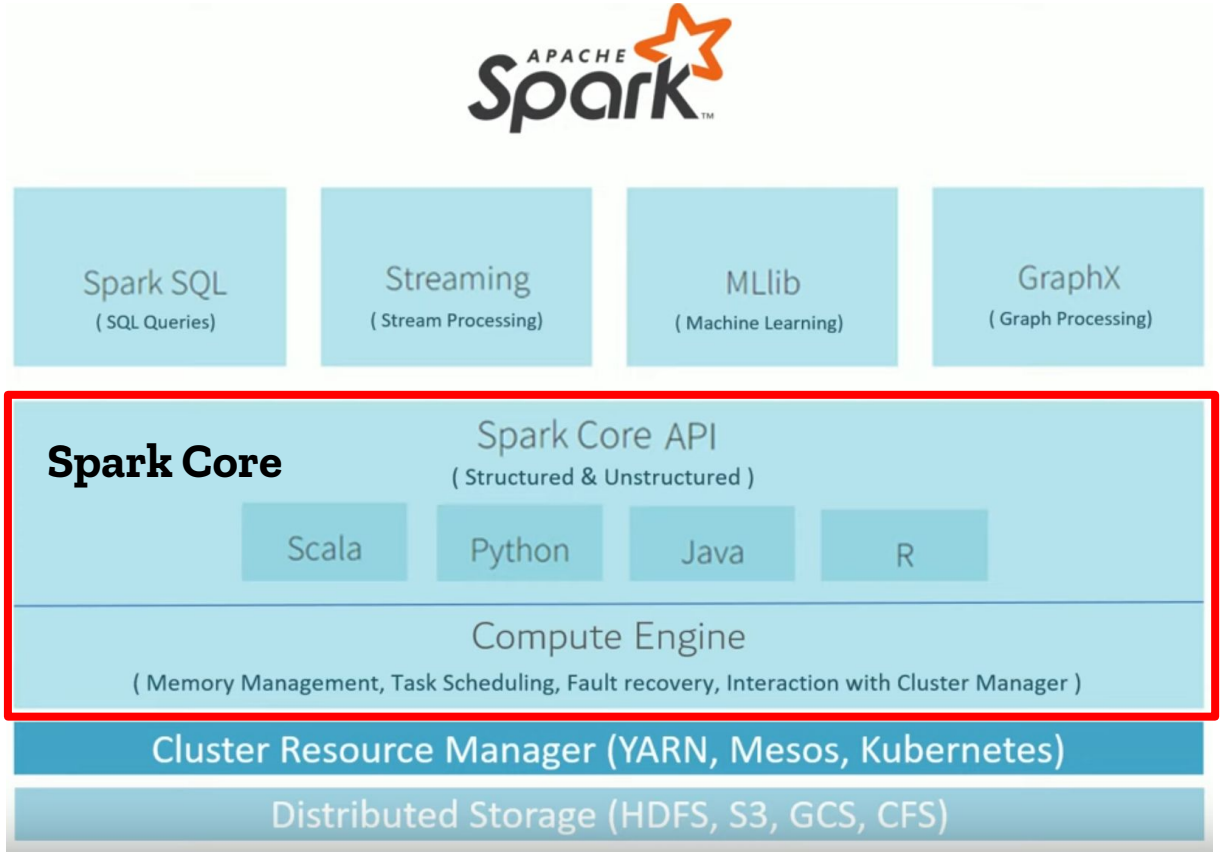
Broadcast variables & Accumulators

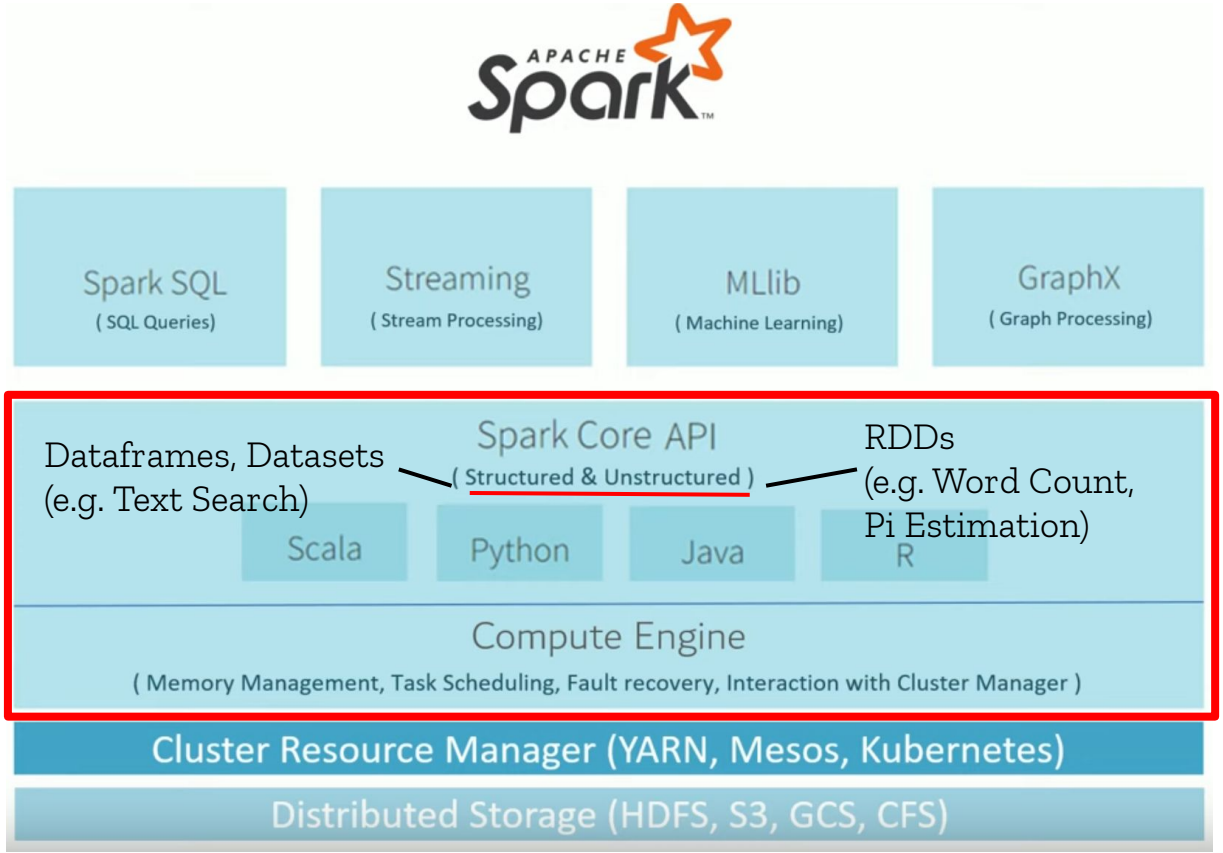
Broadcast Variable

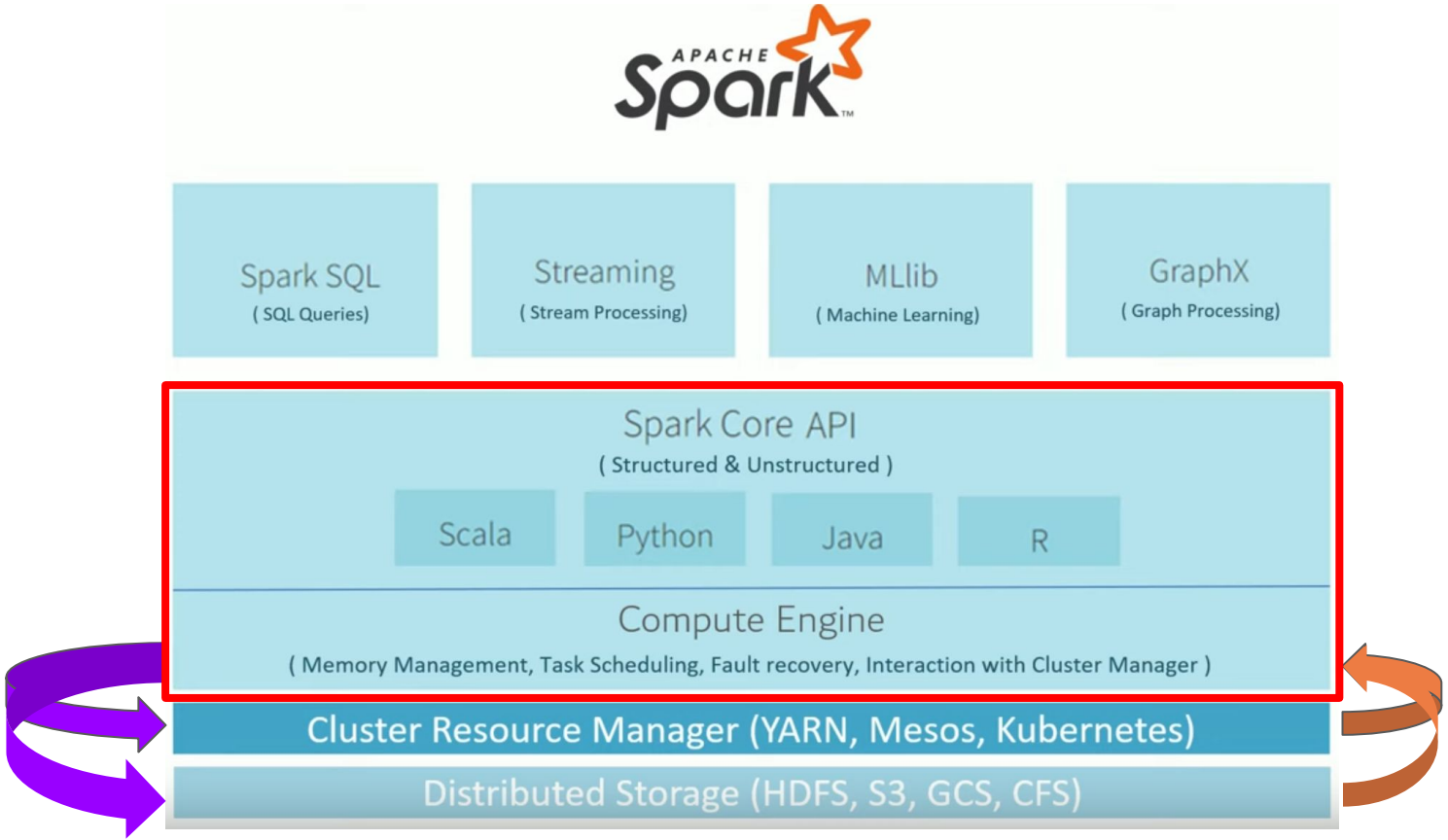


```
val counter = sc.accumulator(0, "counters")
```

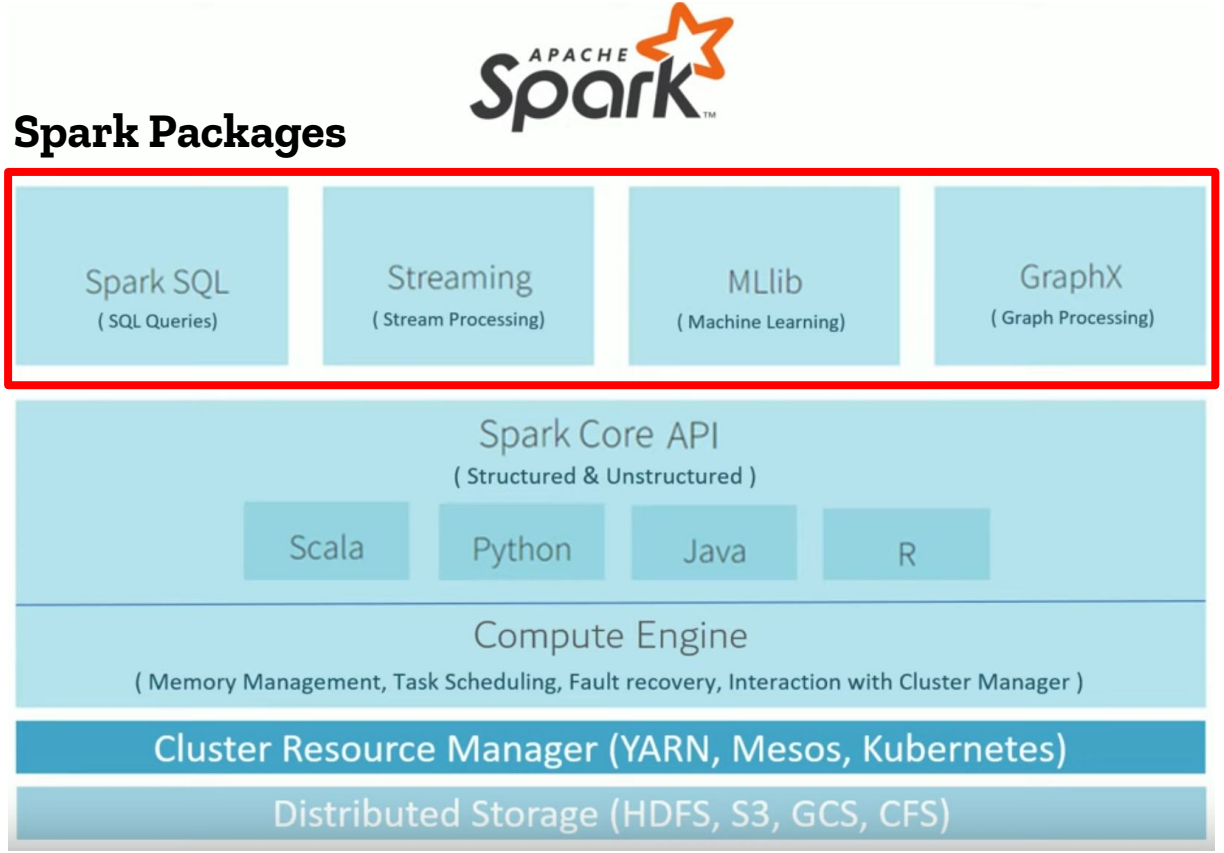

APACHE
Spark
Spark SQL
(SQL Queries)
Streaming
(Stream Processing)
MLlib
(Machine Learning)
GraphX
(Graph Processing)
Spark Core API
(Structured & Unstructured)
Scala
Python
Java
R
Compute Engine
(Memory Management, Task Scheduling, Fault recovery, Interaction with Cluster Manager)
Cluster Resource Manager (YARN, Mesos, Kubernetes)
Distributed Storage (HDFS, S3, GCS, CFS)


APACHE Spark
Spark SQL
(SQL Queries)
Streaming
(Stream Processing)
MLlib
(Machine Learning)
GraphX
(Graph Processing)
Spark Core
Spark Core API
(Structured & Unstructured)
Scala
Python
Java
R
Compute Engine
(Memory Management, Task Scheduling, Fault recovery, Interaction with Cluster Manager)
Cluster Resource Manager (YARN, Mesos, Kubernetes)
Distributed Storage (HDFS, S3, GCS, CFS)


APACHE Spark
Spark SQL
(SQL Queries)
Streaming
(Stream Processing)
MLlib
(Machine Learning)
GraphX
(Graph Processing)
Dataframes, Datasets
(e.g. Text Search)
Spark Core API
(Structured & Unstructured)
RDDs
(e.g. Word Count,
Pi Estimation)
Scala
Python
Java
R
Compute Engine
(Memory Management, Task Scheduling, Fault recovery, Interaction with Cluster Manager)
Cluster Resource Manager (YARN, Mesos, Kubernetes)
Distributed Storage (HDFS, S3, GCS, CFS)


APACHE Spark
Spark SQL
(SQL Queries)
Streaming
(Stream Processing)
MLlib
(Machine Learning)
GraphX
(Graph Processing)
Spark Core API
(Structured & Unstructured)
Scala
Python
Java
R
Compute Engine
(Memory Management, Task Scheduling, Fault recovery, Interaction with Cluster Manager)
Cluster Resource Manager (YARN, Mesos, Kubernetes)
Distributed Storage (HDFS, S3, GCS, CFS)

Spark Packages



Spark Demo

Benchmark Configuration

Keywords

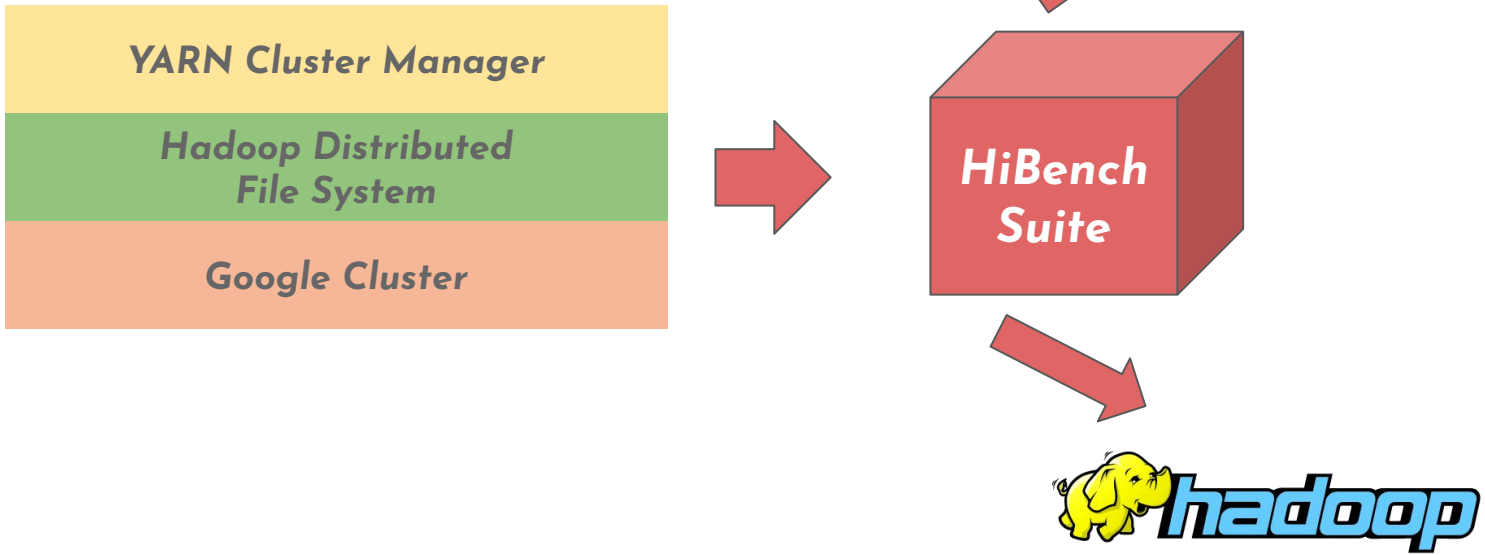
- Google Cloud Platform: Dataproc
- k-Means from Intel HiBench Suite
- Input data points from thousand level to billion level

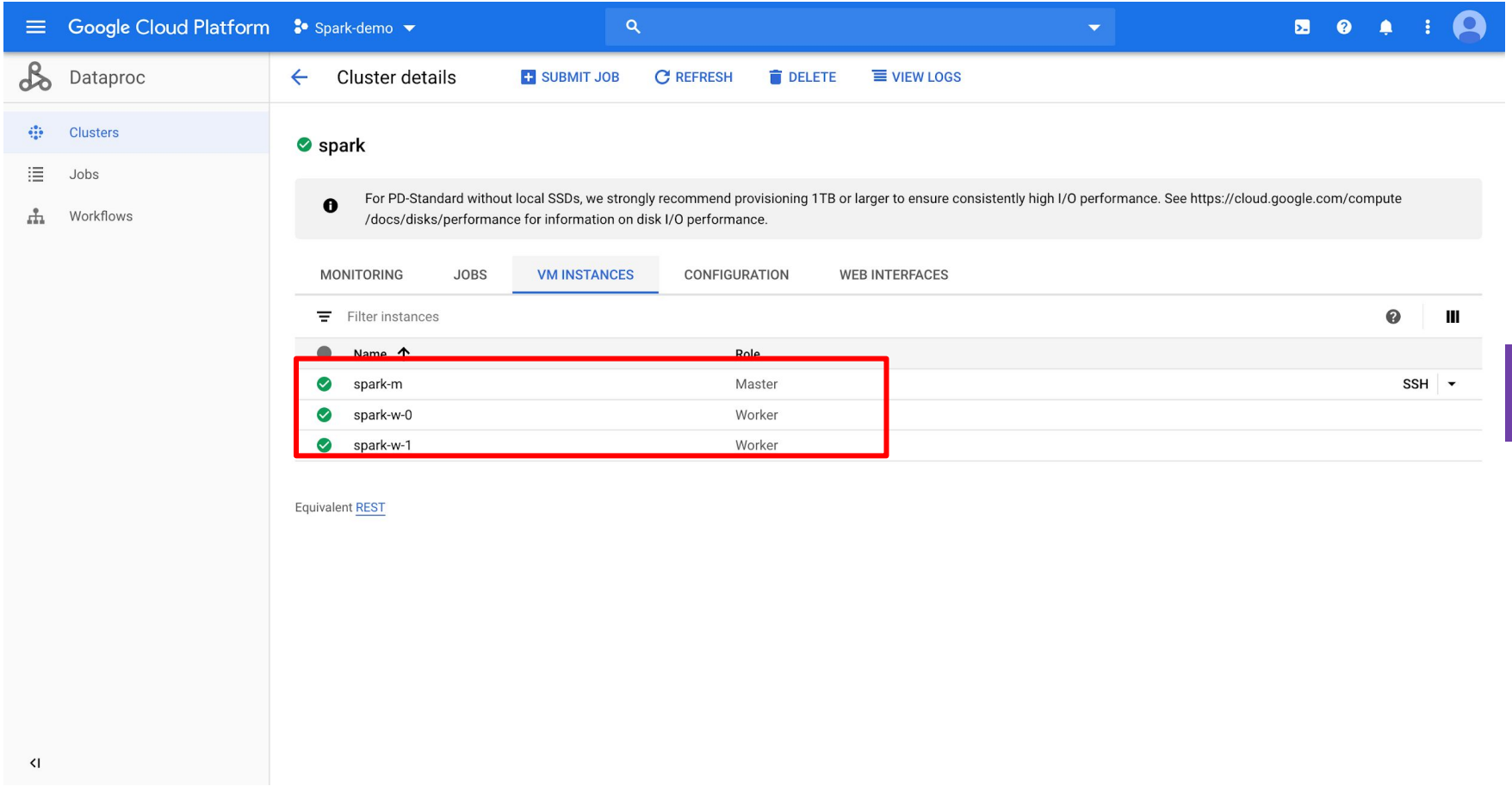
Benchmark Algorithm

K -Means Algorithm with Hadoop + MapReduce vs Spark

- Randomly select k data points as centroids
- Assign input data points to clusters
- Compute new centroids
- Repeat until converges

Benchmark Structure




Google Cloud Platform
Spark-demo
Dataproc
Clusters
Jobs
Workflows
Cluster details
SUBMIT JOB
REFRESH
DELETE
VIEW LOGS
spark
For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consistently high I/O performance. See https://cloud.google.com/compute /docs/disks/performance for information on disk I/O performance.
MONITORING
JOBS
VM INSTANCES
CONFIGURATION
WEB INTERFACES
Filter instances
Name
Role
spark-m
Master
SSH
spark-w-0
Worker
spark-w-1
Worker
Equivalent REST

```
kefan29@spark-m:~/working/HiBench$ bin/workloads/ml/kmeans/hadoop/run.sh
patching args=
Parsing conf: /home/kefan29/working/HiBench/conf/hadoop.conf
Parsing conf: /home/kefan29/working/HiBench/conf/hibench.conf
Parsing conf: /home/kefan29/working/HiBench/conf/spark.conf
Parsing conf: /home/kefan29/working/HiBench/conf/workloads/ml/kmeans.conf
probe sleep jar: /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.9.2.jar
start HadoopKmeans bench
Export env: HADOOP_EXECUTABLE=/usr/lib/hadoop/bin/hadoop
Export env: HADOOP_HOME=/usr/lib/hadoop
Export env: HADOOP_CONF_DIR=/usr/lib/hadoop/etc/hadoop
hdfs rm -r: /usr/lib/hadoop/bin/hadoop --config /usr/lib/hadoop/etc/hadoop fs -rm -r -skipTrash
rm: `hdfs://spark-m:8020/user/kefan29//HiBench/Kmeans/Output': No such file or directory
hdfs du -s: /usr/lib/hadoop/bin/hadoop --config /usr/lib/hadoop/etc/hadoop fs -du -s hdfs://spark-m:8020/user/kefan29//HiBench/Kmeans/Input
20/04/03 23:55:47 INFO Job:  map 45% reduce 0%
```

MapReduce

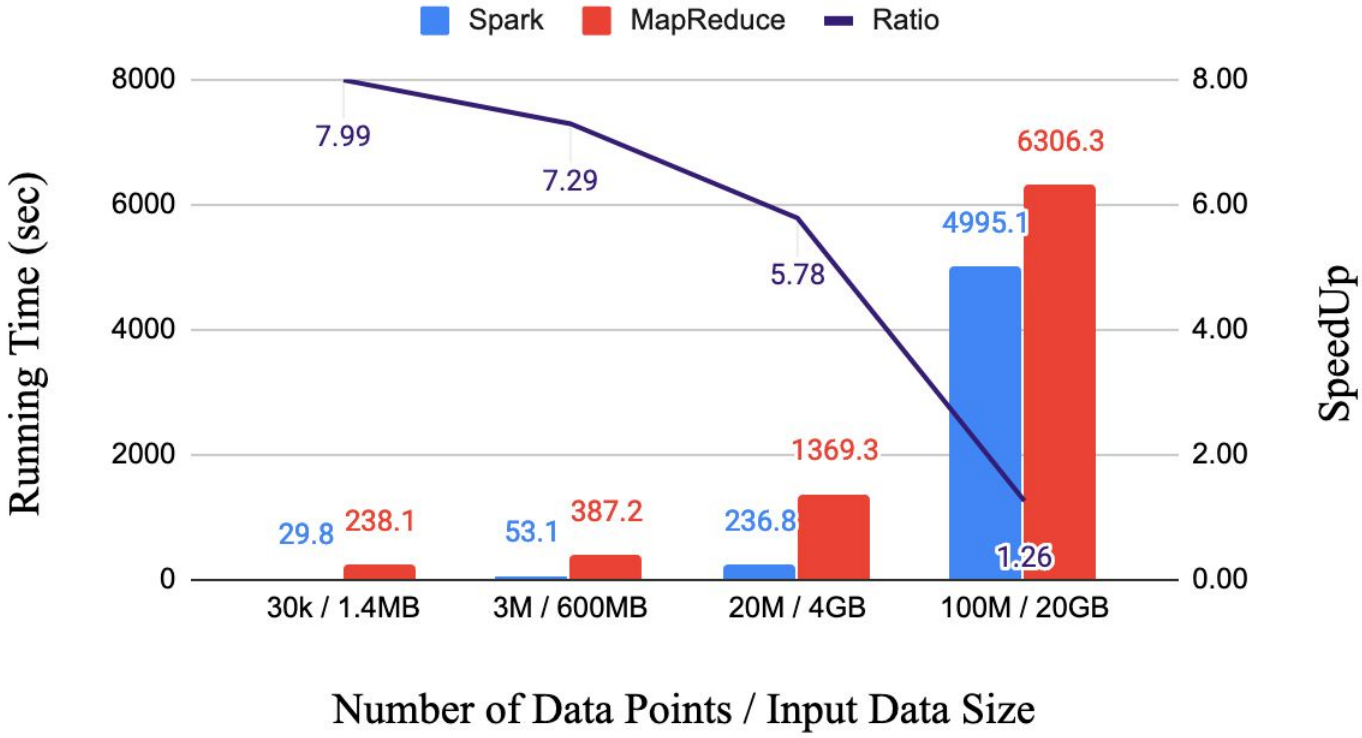
```
kefan29@spark-m:~/working/HiBench$ bin/workloads/ml/kmeans/spark/run.sh
patching args=
Parsing conf: /home/kefan29/working/HiBench/conf/hadoop.conf
Parsing conf: /home/kefan29/working/HiBench/conf/hibench.conf
Parsing conf: /home/kefan29/working/HiBench/conf/spark.conf
Parsing conf: /home/kefan29/working/HiBench/conf/workloads/ml/kmeans.conf
probe sleep jar: /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-2.9.2.jar
start ScalaSparkKmeans bench
hdfs rm -r: /usr/lib/hadoop/bin/hadoop --config /usr/lib/hadoop/etc/hadoop fs
Deleted hdfs://spark-m:8020/user/kefan29/HiBench/Kmeans/Output
hdfs du -s: /usr/lib/hadoop/bin/hadoop --config /usr/lib/hadoop/etc/hadoop fs
Export env: SPARKBENCH_PROPERTIES_FILES=/home/kefan29/working/HiBench/report/
Export env: HADOOP_CONF_DIR=/usr/lib/hadoop/etc/hadoop
Submit Spark job: /usr/lib/spark/bin/spark-submit  --properties-file /home/ke
l.DenseKMeans --master yarn --num-executors 2 --executor-cores 2 --executor-me
 -k 10 --numIterations 5 hdfs://spark-m:8020/user/kefan29//HiBench/Kmeans/Inpu
20/04/03 20:35:18 INFO org.spark_project.jetty.server.AbstractConnector: Stopp
finish ScalaSparkKmeans bench
```

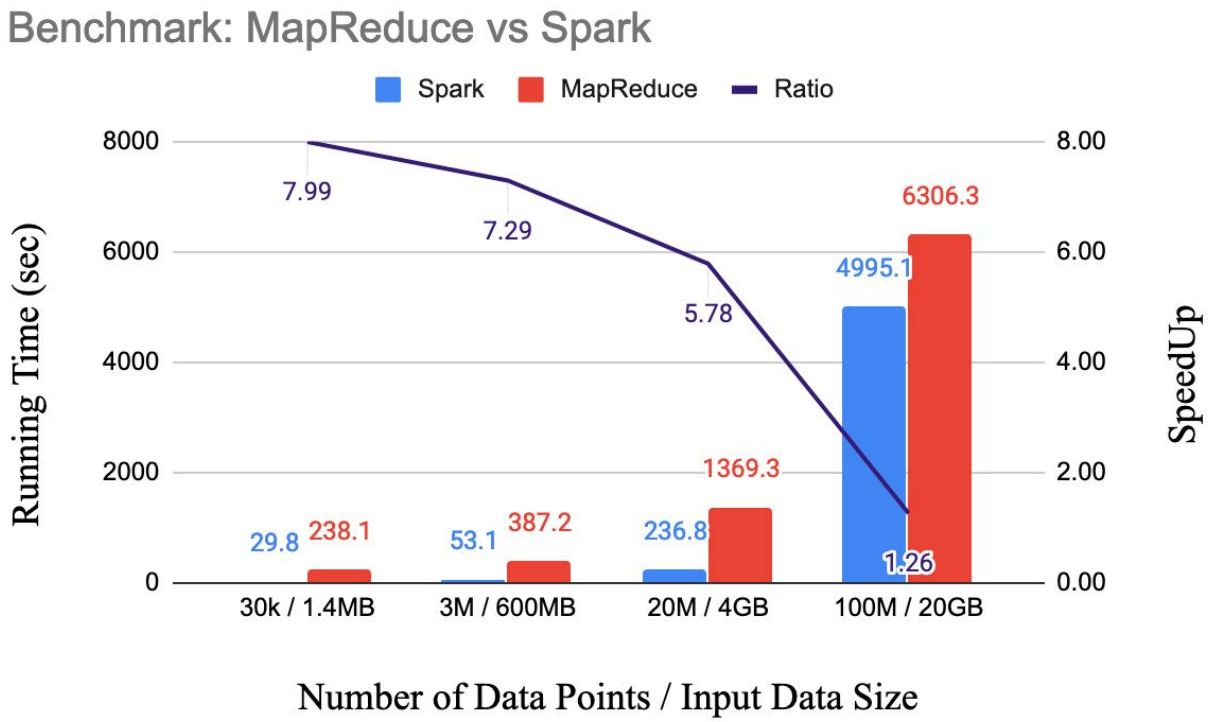
Spark

Benchmark Result

```
kefan29@spark-m:~/working/HiBench/report$ cat hibench.report | column -t
Type              Date        Time      Input_data_size  Duration(s)  Throughput(bytes/s)  Throughput/node
ScalaSparkKmeans  2020-04-03  20:20:52  1396212          30.391       45941                45941
HadoopKmeans      2020-04-03  20:25:48  1396212          238.079      5864                 5864
ScalaSparkKmeans  2020-04-03  20:35:18  602462544        53.126       11340257             11340257
HadoopKmeans      2020-04-03  20:42:36  602462544        387.159      1556111              1556111
ScalaSparkKmeans  2020-04-03  21:04:29  4016371638       777.511      5165678              5165678
HadoopKmeans      2020-04-03  21:28:03  4016371638       1369.341     2933069              2933069
ScalaSparkKmeans  2020-04-03  23:05:45  20081826151      5296.090     3791821              3791821
```

Benchmark: MapReduce vs Spark





Question: Why is there a sudden drop on Spark performance??

Is Memory Size the Bottleneck of Spark Performance? Yes and No

On Apache Spark official website:

Does my data need to fit in memory to use Spark?

No. Spark's operators spill data to disk if it does not fit in memory, allowing it to run well on any sized data. Likewise, cached datasets that do not fit in memory are either spilled to disk or recomputed on the fly when needed, as determined by the RDD's storage level.

From: Does my data need to fit in memory to use Spark?

An Interesting Fact from Another Benchmarking Paper...

"However, the advantage (of Spark) is bounded by the memory. The Speed up goes down when the input is more than 100 million samples and has the minimum value with 1.19x when the input is 400M..."

"...the maximum memory usage for Spark is almost 100 percent with 400M and 800M input that Spark can not create more RDDs at the point. "

From: PERFORMANCE COMPARISON BY RUNNING BENCHMARKS ON HADOOP, SPARK, AND HAMR

...Which would make s

Comparison Between Spark and MapReduce

Advantages and Disadvantages

Comparison between Mapreduce and Spark

	MapReduce	Spark
Code Difficulty	Hard	Easy
Task overhead	Process	Threads
Management	Hard	Easy
Real-time Analysis	Can't Handle	Handle well
Interactive	Can't Handle	Handle well
Streaming	Can't Handle	Handle well
Latency	High	Low
Cost	Low	High

Reference

1. https://data-flair.training/blogs/pyspark-rdd/
2. http://udspace.udel.edu/bitstream/handle/19716/17628/2015_LiuLu_MS.pdf
3. Does my data need to fit in memory to use Spark?
4. https://www.youtube.com/watch?v=QaoJNXW6SQo&list=PL2_PrzItMNs9khZEOnPk8ltMulcYlUyWJ&index=1
5. PERFORMANCE COMPARISON BY RUNNING BENCHMARKS ON HADOOP, SPARK, AND HAMR

Thanks

Any Questions?

However...

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

Resource: Spark officially sets a new record in large-scale sorting

Except for performance reasons..