The Spark Big Data Analytics Platform

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)
1393/10/10

“THAT’S your Ark for the Big Data flood? Noah, you will need a lot more storage space!”

- Big Data refers to datasets and flows large enough that has outpaced our capability to store, process, analyze, and understand.

Where Does Big Data Come From?

Big Data Market Driving Factors

The number of web pages indexed by Google, which were around one million in 1998, have exceeded one trillion in 2008, and its expansion is accelerated by appearance of the social networks.∗

∗"Mining big data: current status, and forecast to the future" [Wei Fan et al., 2013]

Big Data Market Driving Factors

The amount of mobile data traffic is expected to grow to 10.8 Exabyte per month by 2016.∗

∗ "Worldwide Big Data Technology and Services 2012-2015 Forecast" [Dan Vesset et al., 2013]

Big Data Market Driving Factors

More than 65 billion devices were connected to the Internet by 2010, and this number will go up to 230 billion by 2020.∗

∗"The Internet of Things Is Coming" [John Mahoney et al., 2013]

Big Data Market Driving Factors

Many companies are moving towards using Cloud services to access Big Data analytical tools.

Big Data Market Driving Factors

Open source communities

How To Store and Process Big Data?

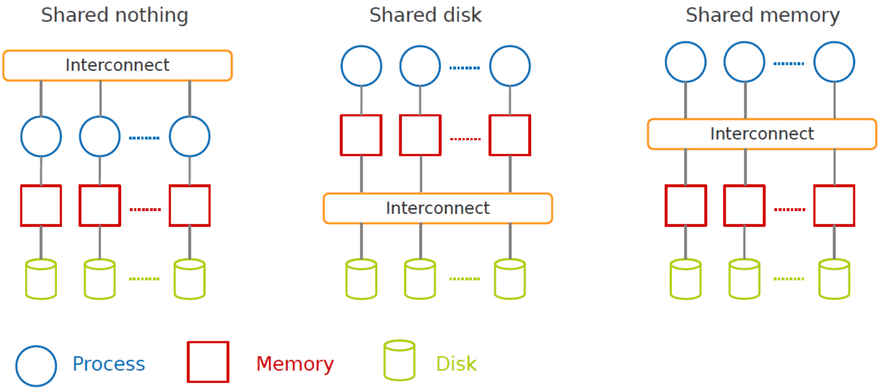
Scale Up vs. Scale Out (1/2)

- Scale up or scale vertically: adding resources to a single node in a system.
- Scale out or scale horizontally: adding more nodes to a system.

Scale Up vs. Scale Out (2/2)

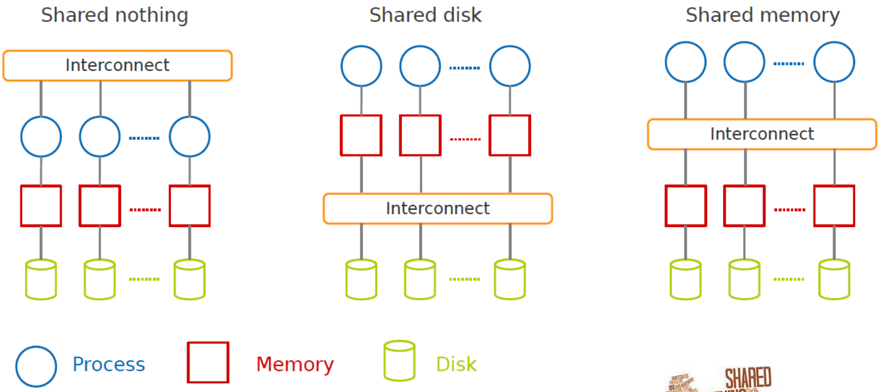
- Scale up: more expensive than scaling out.
- Scale out: more challenging for fault tolerance and software development.

Taxonomy of Parallel Architectures



DeWitt, D. and Gray, J. "Parallel database systems: the future of high performance database systems". ACM Communications, 35(6), 85-98, 1992.

Taxonomy of Parallel Architectures



DeWitt, D. and Gray, J. "Parallel database systems: the future of high performance database systems". ACM Communications, 35(6), 85-98, 1992.

APACHE HBASE

mahout

hadoop

StratoSphere
Above the Clouds

GraphX

Storm

GraphLab

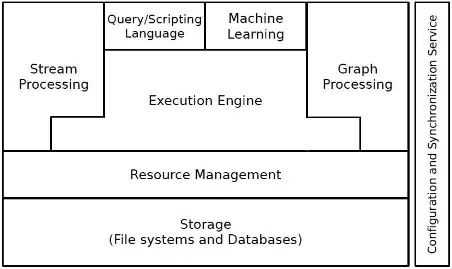
S4 distributed stream computing platform

HIVE

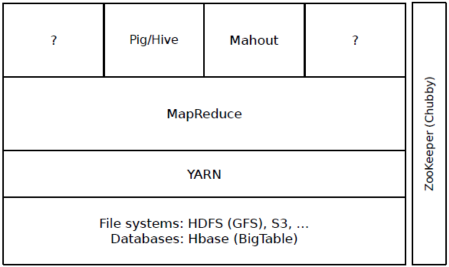
Spark

cassandra

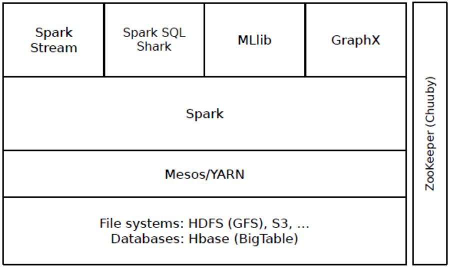
Big Data Analytics Stack



Hadoop Big Data Analytics Stack

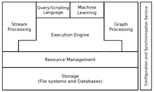


Spark Big Data Analytics Stack



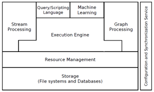
Big Data - File systems

- Traditional file-systems are not well-designed for large-scale data processing systems.



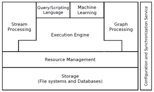
Big Data - File systems

- Traditional file-systems are not well-designed for large-scale data processing systems.
- Efficiency has a higher priority than other features, e.g., directory service.



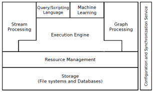
Big Data - File systems

- Traditional file-systems are not well-designed for large-scale data processing systems.
- Efficiency has a higher priority than other features, e.g., directory service.
- Massive size of data tends to store it across multiple machines in a distributed way.



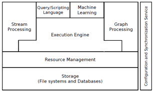
Big Data - File systems

- Traditional file-systems are not well-designed for large-scale data processing systems.
- Efficiency has a higher priority than other features, e.g., directory service.
- Massive size of data tends to store it across multiple machines in a distributed way.
- HDFS/GFS, Amazon S3, ...



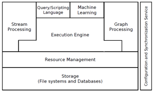
Big Data - Database

- Relational Databases Management Systems (RDMS) were not designed to be distributed.



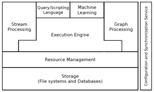
Big Data - Database

- Relational Databases Management Systems (RDMS) were not designed to be distributed.
- NoSQL databases relax one or more of the ACID properties: BASE



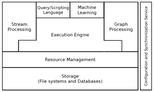
Big Data - Database

- Relational Databases Management Systems (RDMS) were not designed to be distributed.
- NoSQL databases relax one or more of the ACID properties: BASE
- Different data models: key/value, column-family, graph, document.



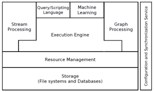
Big Data - Database

- Relational Databases Management Systems (RDMS) were not designed to be distributed.
- NoSQL databases relax one or more of the ACID properties: BASE
- Different data models: key/value, column-family, graph, document.
- Hbase/BigTable, Dynamo, Scalaris, Cassandra, MongoDB, Voldemort, Riak, Neo4J, ...



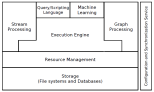
Big Data - Resource Management

- Different frameworks require different computing resources.



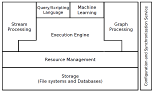
Big Data - Resource Management

- Different frameworks require different computing resources.
- Large organizations need the ability to share data and resources between multiple frameworks.



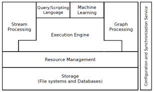
Big Data - Resource Management

- Different frameworks require different computing resources.
- Large organizations need the ability to share data and resources between multiple frameworks.
- Resource management share resources in a cluster between multiple frameworks while providing resource isolation.



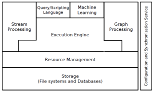
Big Data - Resource Management

- Different frameworks require different computing resources.
- Large organizations need the ability to share data and resources between multiple frameworks.
- Resource management share resources in a cluster between multiple frameworks while providing resource isolation.
- Mesos, YARN, Quincy, ...



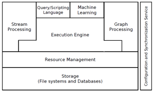
Big Data - Execution Engine

- Scalable and fault tolerance parallel data processing on clusters of unreliable machines.



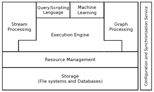
Big Data - Execution Engine

- Scalable and fault tolerance parallel data processing on clusters of unreliable machines.
- Data-parallel programming model for clusters of commodity machines.



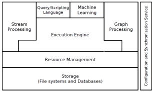
Big Data - Execution Engine

- Scalable and fault tolerance parallel data processing on clusters of unreliable machines.
- Data-parallel programming model for clusters of commodity machines.
- MapReduce, Spark, Stratosphere, Dryad, Hyracks, ...



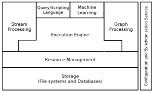
Big Data - Query/Scripting Language

- Low-level programming of execution engines, e.g., MapReduce, is not easy for end users.



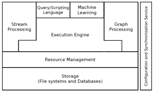
Big Data - Query/Scripting Language

- Low-level programming of execution engines, e.g., MapReduce, is not easy for end users.
- Need high-level language to improve the query capabilities of execution engines.



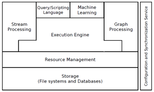
Big Data - Query/Scripting Language

- Low-level programming of execution engines, e.g., MapReduce, is not easy for end users.
- Need high-level language to improve the query capabilities of execution engines.
- It translates user-defined functions to low-level API of the execution engines.



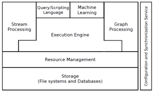
Big Data - Query/Scripting Language

- Low-level programming of execution engines, e.g., MapReduce, is not easy for end users.
- Need high-level language to improve the query capabilities of execution engines.
- It translates user-defined functions to low-level API of the execution engines.
- Pig, Hive, Shark, Meteor, DryadLINQ, SCOPE, ...

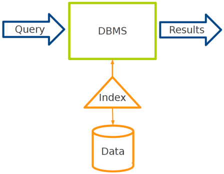


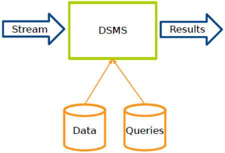
Big Data - Stream Processing

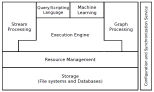
- Providing users with fresh and low latency results.



- Providing users with fresh and low latency results.
- Database Management Systems (DBMS) vs. Data Stream Management Systems (DSMS)

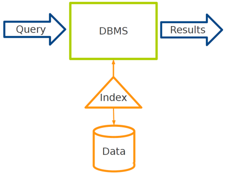


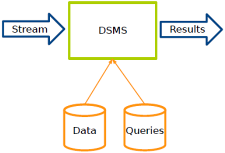




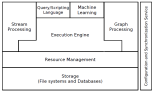
Big Data - Stream Processing

- Providing users with fresh and low latency results.
- Database Management Systems (DBMS) vs. Data Stream Management Systems (DSMS)



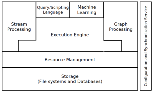


- Storm, S4, SEEP, D-Stream, Naiad, ...



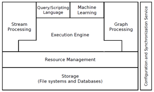
Big Data - Graph Processing

- Many problems are expressed using graphs: sparse computational dependencies, and multiple iterations to converge.



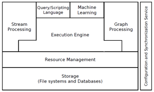
Big Data - Graph Processing

- Many problems are expressed using graphs: sparse computational dependencies, and multiple iterations to converge.
- Data-parallel frameworks, such as MapReduce, are not ideal for these problems: slow



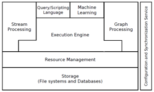
Big Data - Graph Processing

- Many problems are expressed using graphs: sparse computational dependencies, and multiple iterations to converge.
- Data-parallel frameworks, such as MapReduce, are not ideal for these problems: slow
- Graph processing frameworks are optimized for graph-based problems.



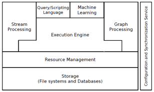
Big Data - Graph Processing

- Many problems are expressed using graphs: sparse computational dependencies, and multiple iterations to converge.
- Data-parallel frameworks, such as MapReduce, are not ideal for these problems: slow
- Graph processing frameworks are optimized for graph-based problems.
- Pregel, Giraph, GraphX, GraphLab, PowerGraph, GraphChi, ...



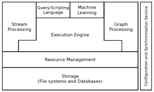
Big Data - Machine Learning

- Implementing and consuming machine learning techniques at scale are difficult tasks for developers and end users.



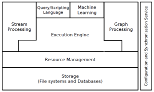
Big Data - Machine Learning

- Implementing and consuming machine learning techniques at scale are difficult tasks for developers and end users.
- There exist platforms that address it by providing scalable machine-learning and data mining libraries.



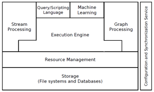
Big Data - Machine Learning

- Implementing and consuming machine learning techniques at scale are difficult tasks for developers and end users.
- There exist platforms that address it by providing scalable machine-learning and data mining libraries.
- Mahout, MLBase, SystemML, Ricardo, Presto, ...



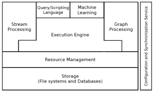
Big Data - Configuration and Synchronization Service

- A means to synchronize distributed applications accesses to shared resources.



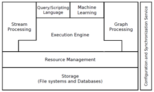
Big Data - Configuration and Synchronization Service

- A means to synchronize distributed applications accesses to shared resources.
- Allows distributed processes to coordinate with each other.



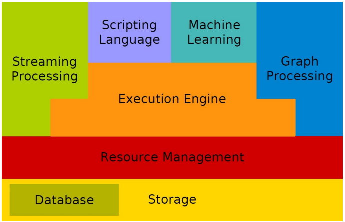
Big Data - Configuration and Synchronization Service

- A means to synchronize distributed applications accesses to shared resources.
- Allows distributed processes to coordinate with each other.
- Zookeeper, Chubby, ...



Outline

- Introduction to HDFS
- Data processing with MapReduce
- Introduction to Scala
- Data exploration using Spark
- Stream processing with Spark Streaming
- Graph analytics with GraphX



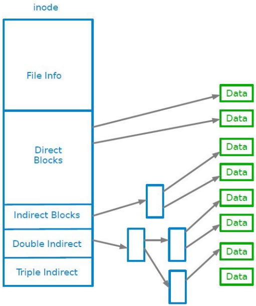
hadoop
HDFS

What is Filesystem?

- Controls how data is stored in and retrieved from disk.

What is Filesystem?

- Controls how data is stored in and retrieved from disk.



Distributed Filesystems

- When data outgrows the storage capacity of a single machine: partition it across a number of separate machines.

- Distributed filesystems: manage the storage across a network of machines.

HDFS

- Hadoop Distributed FileSystem
- Appears as a single disk
- Runs on top of a native filesystem, e.g., ext3
- Fault tolerant: can handle disk crashes, machine crashes, ...
- Based on Google's filesystem GFS

HDFS is Good for ...

- Storing large files
  - Terabytes, Petabytes, etc...
  - 100MB or more per file.
- Streaming data access
  - Data is written once and read many times.
  - Optimized for batch reads rather than random reads.
- Cheap commodity hardware
  - No need for super-computers, use less reliable commodity hardware.

HDFS is Not Good for ...

- Low-latency reads
  - High-throughput rather than low latency for small chunks of data.
  - HBase addresses this issue.

- Large amount of small files
  - Better for millions of large files instead of billions of small files.

- Multiple writers
  - Single writer per file.
  - Writes only at the end of file, no-support for arbitrary offset.

HDFS Daemons (1/2)

- HDFS cluster is manager by three types of processes.

HDFS Daemons (1/2)

- HDFS cluster is manager by three types of processes.

- Namenode
  - Manages the filesystem, e.g., namespace, meta-data, and file blocks
  - Metadata is stored in memory.

HDFS Daemons (1/2)

- HDFS cluster is manager by three types of processes.

- Namenode
  - Manages the filesystem, e.g., namespace, meta-data, and file blocks
  - Metadata is stored in memory.

- Datanode
  - Stores and retrieves data blocks
  - Reports to Namenode
  - Runs on many machines

HDFS Daemons (1/2)

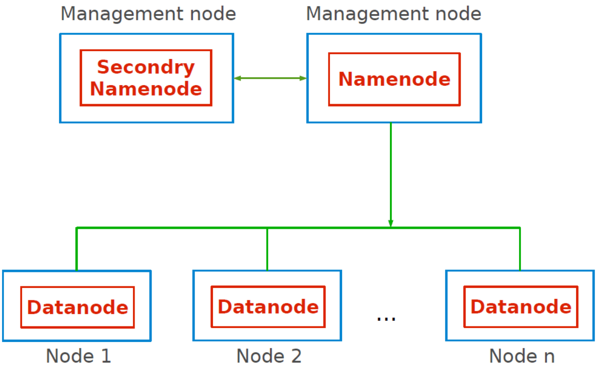
- HDFS cluster is manager by three types of processes.

- Namenode
  - Manages the filesystem, e.g., namespace, meta-data, and file blocks
  - Metadata is stored in memory.

- Datanode
  - Stores and retrieves data blocks
  - Reports to Namenode
  - Runs on many machines

- Secondary Namenode
  - Only for checkpointing.
  - Not a backup for Namenode

HDFS Daemons (2/2)



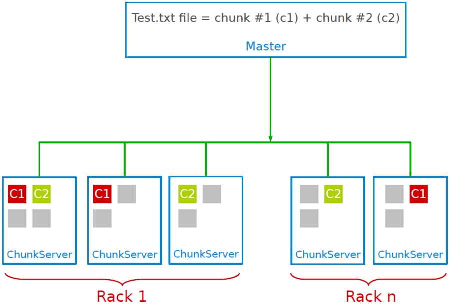
Files and Blocks (1/2)

- Files are split into blocks.
- Blocks
  - Single unit of storage: a contiguous piece of information on a disk.
  - Transparent to user.
  - Managed by Namenode, stored by Datanode.
  - Blocks are traditionally either 64MB or 128MB: default is 64MB.

Files and Blocks (2/2)

- Same block is replicated on multiple machines: default is 3
  - Replica placements are rack aware.
  - 1st replica on the local rack.
  - 2nd replica on the local rack but different machine.
  - 3rd replica on the different rack.

- Namenode determines replica placement.

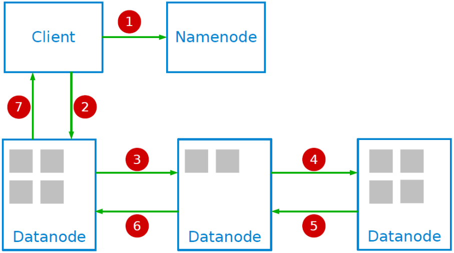


HDFS Client

- Client interacts with Namenode
  - To update the Namenode namespace.
  - To retrieve block locations for writing and reading.
- Client interacts directly with Datanode
  - To read and write data.
- Namenode does not directly write or read data.

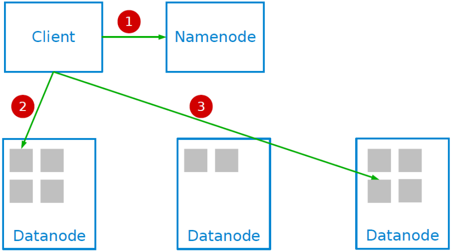
HDFS Write

- 1. Create a new file in the Namenode's Namespace; calculate block topology.
- 2, 3, 4. Stream data to the first, second and third node.
- 5, 6, 7. Success/failure acknowledgment.



HDFS Read

- 1. Retrieve block locations.
- 2, 3. Read blocks to re-assemble the file.



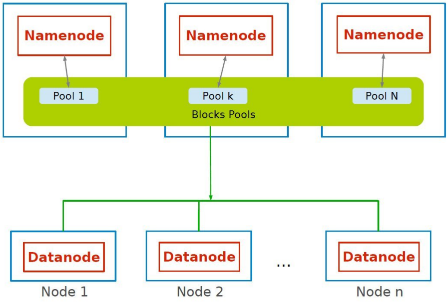
Namenode Memory Concerns

- For fast access Namenode keeps all block metadata in-memory.
  - Will work well for clusters of 100 machines.

- Changing block size will affect how much space a cluster can host.
  - 64MB to 128MB will reduce the number of blocks and increase the space that Namenode can support.

HDFS Federation

- Hadoop 2+
- Each Namenode will host part of the blocks.
- A Block Pool is a set of blocks that belong to a single namespace.
- Support for 1000+ machine clusters.



Namenode Fault-Tolerance (1/2)

- Namenode is a single point of failure.

- If Namenode crashes then cluster is down.

Namenode Fault-Tolerance (1/2)

- Namenode is a single point of failure.
- If Namenode crashes then cluster is down.
- Secondary Namenode periodically merges the namespace image and log and a persistent record of it written to disk (checkpointing).
- But, the state of the secondary Namenode lags that of the primary: does not provide high-availability of the filesystem

Namenode Fault-Tolerance (2/2)

- High availability Namenode.
  - Hadoop 2+
  - Active standby is always running and takes over in case main Namenode fails.

HDFS Installation and Shell

HDFS Installation

- Three options
  - Local (Standalone) Mode
  - Pseudo-Distributed Mode
  - Fully-Distributed Mode

Installation - Local

- Default configuration after the download.
- Executes as a single Java process.
- Works directly with local filesystem.
- Useful for debugging.

Installation - Pseudo-Distributed (1/6)

- Still runs on a single node.
- Each daemon runs in its own Java process.
  - Namenode
  - Secondary Namenode
  - Datanode
- Configuration files:
  - `hadoop-env.sh`
  - `core-site.xml`
  - `hdfs-site.xml`

Installation - Pseudo-Distributed (2/6)

- Specify environment variables in `hadoop-env.sh`

```
export JAVA_HOME=/opt/jdk1.7.0_51
```

Installation - Pseudo-Distributed (3/6)

- Specify location of Namenode in `core-site.sh`

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:8020</value>
  <description>NameNode URI</description>
</property>
```

Installation - Pseudo-Distributed (4/6)

- Configurations of Namenode in `hdfs-site.sh`
- Path on the local filesystem where the Namenode stores the namespace and transaction logs persistently.

```
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/opt/hadoop-2.2.0/hdfs/namenode</value>
  <description>description...</description>
</property>
```

Installation - Pseudo-Distributed (5/6)

- Configurations of Secondary Namenode in `hdfs-site.sh`
- Path on the local filesystem where the Secondary Namenode stores the temporary images to merge.

```
<property>
  <name>dfs.namenode.checkpoint.dir</name>
  <value>/opt/hadoop-2.2.0/hdfs/secondary</value>
  <description>description...</description>
</property>
```

Installation - Pseudo-Distributed (6/6)

- Configurations of Datanode in `hdfs-site.sh`
- Comma separated list of paths on the local filesystem of a Datanode where it should store its blocks.

```
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/opt/hadoop-2.2.0/hdfs/datanode</value>
  <description>description...</description>
</property>
```

Start HDFS and Test

- Format the Namenode directory (do this only once, the first time).

```
hdfs namenode -format
```

Start HDFS and Test

- Format the Namenode directory (do this only once, the first time).

```
hdfs namenode -format
```

- Start the Namenode, Secondary namenode and Datanode daemons.

```
hadoop-daemon.sh start namenode
hadoop-daemon.sh start secondarynamenode
hadoop-daemon.sh start datanode
jps
```

Start HDFS and Test

- Format the Namenode directory (do this only once, the first time).

```
hdfs namenode -format
```

- Start the Namenode, Secondary namenode and Datanode daemons.

```
hadoop-daemon.sh start namenode
hadoop-daemon.sh start secondarynamenode
hadoop-daemon.sh start datanode
jps
```

- Verify the deamons are running:
  - Namenode: `http://localhost:50070`
  - Secondary Namenode: `http://localhost:50090`
  - Datanode: `http://localhost:50075`

HDFS Shell

```
hdfs dfs -<command> -<option> <path>
```

HDFS Shell

```
hdfs dfs -<command> -<option> <path>
```

```
hdfs dfs -ls /
hdfs dfs -ls file:///home/big
hdfs dfs -ls hdfs://localhost/
hdfs dfs -cat /dir/file.txt
hdfs dfs -cp /dir/file1 /otherDir/file2
hdfs dfs -mv /dir/file1 /dir2/file2
hdfs dfs -mkdir /newDir
hdfs dfs -put file.txt /dir/file.txt # can also use copyFromLocal
hdfs dfs -get /dir/file.txt file.txt # can also use copyToLocal
hdfs dfs -rm /dir/fileToDelete
hdfs dfs -help
```

MapReduce

- A shared nothing architecture for processing large data sets with a parallel/distributed algorithm on clusters.

MapReduce Definition

- A programming model: to batch process large data sets (inspired by functional programming).

MapReduce Definition

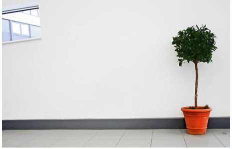
- A programming model: to batch process large data sets (inspired by functional programming).

- An execution framework: to run parallel algorithms on clusters of commodity hardware.

Simplicity

- Don't worry about parallelization, fault tolerance, data distribution, and load balancing (MapReduce takes care of these).
- Hide system-level details from programmers.

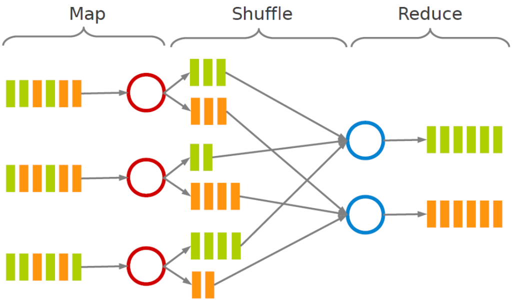
Simplicity!



Programming Model

MapReduce Dataflow

- map function: processes data and generates a set of intermediate key/value pairs.
- reduce function: merges all intermediate values associated with the same intermediate key.



Example: Word Count

- Consider doing a word count of the following file using MapReduce:

```
Hello World Bye World
Hello Hadoop Goodbye Hadoop
```

Example: Word Count - map

- The map function reads in words one a time and outputs (word, 1) for each parsed input word.

- The map function output is:

```
(Hello, 1)
(World, 1)
(Bye, 1)
(World, 1)
(Hello, 1)
(Hadoop, 1)
(Goodbye, 1)
(Hadoop, 1)
```

Example: Word Count - shuffle

- The shuffle phase between map and reduce phase creates a list of values associated with each key.

- The reduce function input is:

```
(Bye, (1))
(Goodbye, (1))
(Hadoop, (1, 1)
(Hello, (1, 1))
(World, (1, 1))
```

Example: Word Count - reduce

- The reduce function sums the numbers in the list for each key and outputs (word, count) pairs.
- The output of the reduce function is the output of the MapReduce job:

```
(Bye, 1)
(Goodbye, 1)
(Hadoop, 2)
(Hello, 2)
(World, 2)
```

Combiner Function (1/2)

- In some cases, there is significant repetition in the intermediate keys produced by each map task, and the reduce function is commutative and associative.

Machine 1:

```
(Hello, 1)
(World, 1)
(Bye, 1)
(World, 1)
```

Machine 2:

```
(Hello, 1)
(Hadoop, 1)
(Goodbye, 1)
(Hadoop, 1)
```

Combiner Function (2/2)

- Users can specify an optional combiner function to merge partially data before it is sent over the network to the reduce function.
- Typically the same code is used to implement both the combiner and the reduce function.

Machine 1:

```
(Hello, 1)
(World, 2)
(Bye, 1)
```

Machine 2:

```
(Hello, 1)
(Hadoop, 2)
(Goodbye, 1)
```

Example: Word Count - map

```
public static class MyMap extends Mapper<...> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);

    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }
  }
}
```

Example: Word Count - reduce

```
public static class MyReduce extends Reducer<...> {
  public void reduce(Text key, Iterator<...> values, Context context)
    throws IOException, InterruptedException {
    int sum = 0;

    while (values.hasNext())
      sum += values.next().get();

    context.write(key, new IntWritable(sum));
  }
}
```

Example: Word Count - driver

```java
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = new Job(conf, "wordcount");

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  job.setMapperClass(MyMap.class);
  job.setCombinerClass(MyReduce.class);
  job.setReducerClass(MyReduce.class);

  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);

  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  job.waitForCompletion(true);
}
```

Example: Word Count - Compile and Run (1/2)

```
# start hdfs
> hadoop-daemon.sh start namenode
> hadoop-daemon.sh start datanode

# make the input folder in hdfs
> hdfs dfs -mkdir -p input

# copy input files from local filesystem into hdfs
> hdfs dfs -put file0 input/file0
> hdfs dfs -put file1 input/file1

> hdfs dfs -ls input/
input/file0
input/file1

> hdfs dfs -cat input/file0
Hello World Bye World

> hdfs dfs -cat input/file1
Hello Hadoop Goodbye Hadoop
```

Example: Word Count - Compile and Run (2/2)

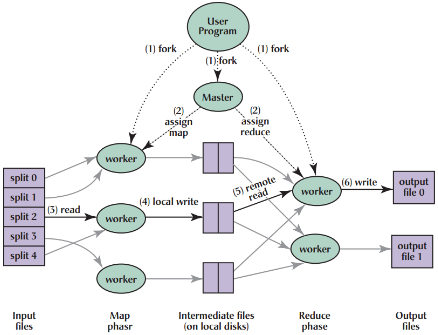
```
> mkdir wordcount_classes

> javac -classpath
$HADOOP_HOME/share/hadoop/common/hadoop-common-2.2.0.jar:
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.2.0.jar:
$HADOOP_HOME/share/hadoop/common/lib/commons-cli-1.2.jar
-d wordcount_classes sics/WordCount.java

> jar -cvf wordcount.jar -C wordcount_classes/ .

> hadoop jar wordcount.jar sics.WordCount input output

> hdfs dfs -ls output
output/part-00000

> hdfs dfs -cat output/part-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

Execution Engine

MapReduce Execution (1/7)

- The user program divides the input files into M splits.
  - A typical size of a split is the size of a HDFS block (64 MB).
  - Converts them to key/value pairs.

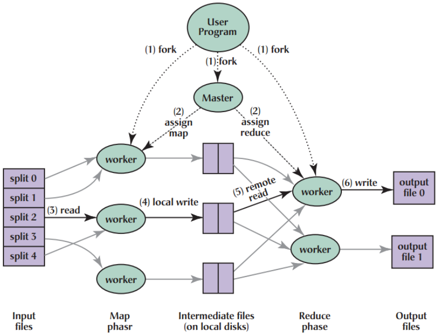
- It starts up many copies of the program on a cluster of machines.



J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", ACM Communications 51(1), 2008.

MapReduce Execution (2/7)

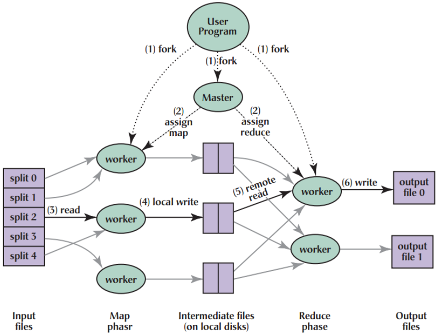
- One of the copies of the program is master, and the rest are workers.
- The master assigns works to the workers.
  - It picks idle workers and assigns each one a map task or a reduce task.



J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", ACM Communications 51(1), 2008.

MapReduce Execution (3/7)

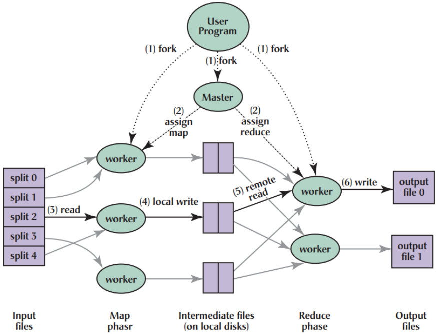
- A map worker reads the contents of the corresponding input splits.
- It parses key/value pairs out of the input data and passes each pair to the user defined map function.
- The intermediate key/value pairs produced by the map function are buffered in memory.



J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", ACM Communications 51(1), 2008.

MapReduce Execution (4/7)

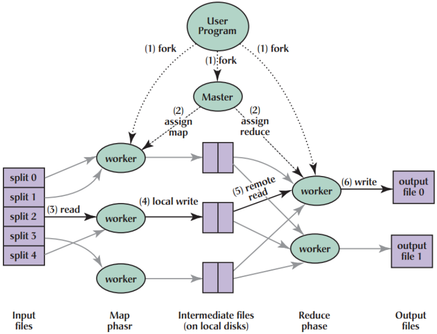
- The buffered pairs are periodically written to local disk.
  - They are partitioned into R regions (hash(key) mod R).
- The locations of the buffered pairs on the local disk are passed back to the master.
- The master forwards these locations to the reduce workers.



J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters”, ACM Communications 51(1), 2008.

MapReduce Execution (5/7)

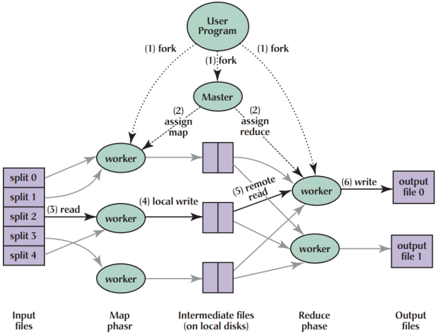
- A reduce worker reads the buffered data from the local disks of the map workers.
- When a reduce worker has read all intermediate data, it sorts it by the intermediate keys.



J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", ACM Communications 51(1), 2008.

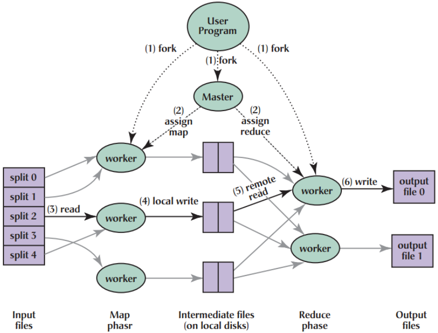
MapReduce Execution (6/7)

- The reduce worker iterates over the intermediate data.
- For each unique intermediate key, it passes the key and the corresponding set of intermediate values to the user defined reduce function.
- The output of the reduce function is appended to a final output file for this reduce partition.



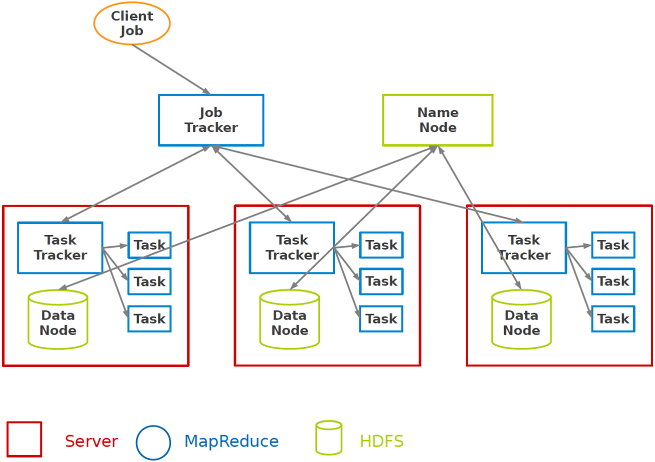
MapReduce Execution (7/7)

- When all map tasks and reduce tasks have been completed, the master wakes up the user program.



J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", ACM Communications 51(1), 2008.

Hadoop MapReduce and HDFS



Fault Tolerance

- On worker failure:
  - Detect failure via periodic heartbeats.
  - Re-execute in-progress map and reduce tasks.
  - Re-execute completed map tasks: their output is stored on the local disk of the failed machine and is therefore inaccessible.
  - Completed reduce tasks do not need to be re-executed since their output is stored in a global filesystem.

- On master failure:
  - State is periodically checkpointed: a new copy of master starts from the last checkpoint state.

Scala

- Scala: scalable language
- A blend of object-oriented and functional programming
- Runs on the Java Virtual Machine
- Designed by Martin Odersky at EPFL

Functional Programming Languages

- In a restricted sense: a language that does not have mutable variables, assignments, or imperative control structures.

- In a wider sense: it enables the construction of programs that focus on functions.

Functional Programming Languages

- In a restricted sense: a language that does not have mutable variables, assignments, or imperative control structures.

- In a wider sense: it enables the construction of programs that focus on functions.

- Functions are first-class citizens:
  - Defined anywhere (including inside other functions).
  - Passed as parameters to functions and returned as results.
  - Operators to compose functions.

Scala Variables

- Values: immutable
- Variables: mutable

```scala
var myVar: Int = 0
val myVal: Int = 1
```

- Scala data types:
  - Boolean, Byte, Short, Char, Int, Long, Float, Double, String

If ... Else

```
var x = 30;

if (x == 10) {
  println("Value of X is 10");
} else if (x == 20) {
  println("Value of X is 20");
} else {
  println("This is else statement");
}
```

Loop

```
var a = 0
var b = 0
for (a <- 1 to 3; b <- 1 until 3) {
  println("Value of a: " + a + ", b: " + b )
}
```

```
// loop with collections
val numList = List(1, 2, 3, 4, 5, 6)
for (a <- numList) {
  println("Value of a: " + a)
}
```

Functions

```
def functionName([list of parameters]): [return type] = {
  function body
  return [expr]
}

def addInt(a: Int, b: Int): Int = {
  var sum: Int = 0
  sum = a + b
  sum
}

println("Returned Value: " + addInt(5, 7))
```

Anonymous Functions

- Lightweight syntax for defining functions.

```
var mul = (x: Int, y: Int) => x * y
println(mul(3, 4))
```

Higher-Order Functions


```
def apply(f: Int => String, v: Int) = f(v)

def layout(x: Int) = "[" + x.toString() + "]"

println(apply(layout, 10))
```

Collections (1/2)

- Array: fixed-size sequential collection of elements of the same type

```
val t = Array("zero", "one", "two")
val b = t(0) // b = zero
```

Collections (1/2)

- **Array**: fixed-size sequential collection of elements of the same type

```
val t = Array("zero", "one", "two")
val b = t(0) // b = zero
```

- **List**: sequential collection of elements of the same type

```
val t = List("zero", "one", "two")
val b = t(0) // b = zero
```

Collections (1/2)

- Array: fixed-size sequential collection of elements of the same type

```
val t = Array("zero", "one", "two")
val b = t(0) // b = zero
```

- List: sequential collection of elements of the same type

```
val t = List("zero", "one", "two")
val b = t(0) // b = zero
```

- Set: sequential collection of elements of the same type without duplicates

```
val t = Set("zero", "one", "two")
val t.contains("zero")
```

Collections (2/2)

- Map: collection of key/value pairs

```
val m = Map(1 -> "sics", 2 -> "kth")
val b = m(1) // b = sics
```

Collections (2/2)

- Map: collection of key/value pairs

```
val m = Map(1 -> "sics", 2 -> "kth")
val b = m(1) // b = sics
```

- Tuple: A fixed number of items of different types together

```
val t = (1, "hello")
val b = t._1 // b = 1
val c = t._2 // c = hello
```

Functional Combinators

- map: applies a function over each element in the list

```
val numbers = List(1, 2, 3, 4)
numbers.map(i => i * 2) // List(2, 4, 6, 8)
```

- flatten: it collapses one level of nested structure

```
List(List(1, 2), List(3, 4)).flatten // List(1, 2, 3, 4)
```

- flatMap: map + flatten

- foreach: it is like map but returns nothing

Classes and Objects

```
class Calculator {
  val brand: String = "HP"
  def add(m: Int, n: Int): Int = m + n
}

val calc = new Calculator
calc.add(1, 2)
println(calc.brand)
```

Classes and Objects

```scala
class Calculator {
  val brand: String = "HP"
  def add(m: Int, n: Int): Int = m + n
}

val calc = new Calculator
calc.add(1, 2)
println(calc.brand)
```

- A singleton is a class that can have only one instance.

```scala
object Test {
  def main(args: Array[String]) { ... }
}

Test.main(null)
```

Case Classes and Pattern Matching

- Case classes are used to store and match on the contents of a class.
- They are designed to be used with pattern matching.
- You can construct them without using new.

```
case class Calc(brand: String, model: String)

def calcType(calc: Calc) = calc match {
  case Calc("hp", "20B") => "financial"
  case Calc("hp", "48G") => "scientific"
  case Calc("hp", "30B") => "business"
  case _ => "Calculator of unknown type"
}

calcType(Calc("hp", "20B"))
```

Simple Build Tool (SBT)

- An open source build tool for Scala and Java projects.
- Similar to Java's Maven or Ant.
- It is written in Scala.

SBT - Hello World!

```scala
// make dir hello and edit Hello.scala
object Hello {
  def main(args: Array[String]) {
    println("Hello world.")
  }
}
```

```
$ cd hello
$ sbt compile run
```

Common Commands

- `compile`: compiles the main sources.
- `run <argument>*`: run the main class.
- `package`: creates a jar file.
- `console`: starts the Scala interpreter.
- `clean`: deletes all generated files.
- `help <command>`: displays detailed help for the specified command.

Create a Simple Project

- Create `project` directory.
- Create `src/main/scala` directory.
- Create `build.sbt` in the project root.

build.sbt

- A list of Scala expressions, separated by blank lines.
- Located in the project's base directory.

```
$ cat build.sbt
name := "hello"

version := "1.0"

scalaVersion := "2.10.4"
```

Add Dependencies

- Add in `build.sbt`.
- Module ID format:
  `"groupID" %% "artifact" % "version" % "configuration"`

```
libraryDependencies += "org.apache.spark" %% "spark-core" % "1.0.0"

// multiple dependencies
libraryDependencies ++= Seq(
    "org.apache.spark" %% "spark-core" % "1.0.0",
    "org.apache.spark" %% "spark-streaming" % "1.0.0"
)
```

- sbt uses the standard Maven2 repository by default, but you can add more resolvers.

```
resolvers += "Akka Repository" at "http://repo.akka.io/releases/"
```

Scala Hands-on Exercises (1/3)

- Declare a list of integers as a variable called `myNumbers`

Scala Hands-on Exercises (1/3)

- Declare a list of integers as a variable called `myNumbers`

```
val myNumbers = List(1, 2, 5, 4, 7, 3)
```

Scala Hands-on Exercises (1/3)

- Declare a list of integers as a variable called `myNumbers`

```
val myNumbers = List(1, 2, 5, 4, 7, 3)
```

- Declare a function, `pow`, that computes the second power of an Int

Scala Hands-on Exercises (1/3)

- Declare a list of integers as a variable called `myNumbers`

```
val myNumbers = List(1, 2, 5, 4, 7, 3)
```

- Declare a function, `pow`, that computes the second power of an Int

```
def pow(a: Int): Int = a * a
```

Scala Hands-on Exercises (2/3)

- Apply the function to `myNumbers` using the `map` function

Scala Hands-on Exercises (2/3)

- Apply the function to `myNumbers` using the `map` function

```
myNumbers.map(x => pow(x))
// or
myNumbers.map(pow(_))
// or
myNumbers.map(pow)
```

Scala Hands-on Exercises (2/3)

- Apply the function to `myNumbers` using the `map` function

```
myNumbers.map(x => pow(x))
// or
myNumbers.map(pow(_))
// or
myNumbers.map(pow)
```

- Write the `pow` function inline in a `map` call, using closure notation

Scala Hands-on Exercises (2/3)

- Apply the function to `myNumbers` using the `map` function

```
myNumbers.map(x => pow(x))
// or
myNumbers.map(pow(_))
// or
myNumbers.map(pow)
```

- Write the `pow` function inline in a `map` call, using closure notation

```
myNumbers.map(x => x * x)
```

Scala Hands-on Exercises (2/3)

- Apply the function to myNumbers using the map function

```
myNumbers.map(x => pow(x))
// or
myNumbers.map(pow(_))
// or
myNumbers.map(pow)
```

- Write the pow function inline in a map call, using closure notation

```
myNumbers.map(x => x * x)
```

- Iterate through myNumbers and print out its items

Scala Hands-on Exercises (2/3)

- Apply the function to `myNumbers` using the `map` function

```
myNumbers.map(x => pow(x))
// or
myNumbers.map(pow(_))
// or
myNumbers.map(pow)
```

- Write the `pow` function inline in a `map` call, using closure notation

```
myNumbers.map(x => x * x)
```

- Iterate through `myNumbers` and print out its items

```
for (i <- myNumbers)
  println(i)
// or
myNumbers.foreach(println)
```

Scala Hands-on Exercises (3/3)

- Declare a list of pair of string and integers as a variable called `myList`

Scala Hands-on Exercises (3/3)

- Declare a list of pair of string and integers as a variable called `myList`

```
val myList = List[(String, Int)](("a", 1), ("b", 2), ("c", 3))
```

Scala Hands-on Exercises (3/3)

- Declare a list of pair of string and integers as a variable called `myList`

```
val myList = List[(String, Int)](("a", 1), ("b", 2), ("c", 3))
```

- Write an inline function to increment the integer values of the list `myList`

Scala Hands-on Exercises (3/3)

- Declare a list of pair of string and integers as a variable called `myList`

```
val myList = List[(String, Int)](("a", 1), ("b", 2), ("c", 3))
```

- Write an inline function to increment the integer values of the list `myList`

```
val x = v.map { case (name, age) =>  age + 1 }
// or
val x = v.map(i => i._2 + 1)
// or
val x = v.map(_._2 + 1)
```

What is Spark?

- An efficient distributed general-purpose data analysis platform.
- Focusing on ease of programming and high performance.

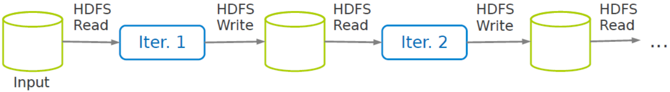
Motivation

- MapReduce programming model has not been designed for complex operations, e.g., data mining.
- Very expensive, i.e., always goes to disk and HDFS.

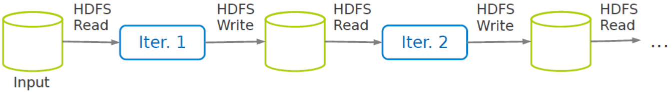
Solution

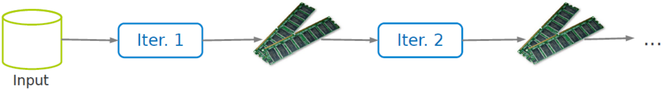
- Extends MapReduce with more operators.
- Support for advanced data flow graphs.
- In-memory and out-of-core processing.

Spark vs. Hadoop

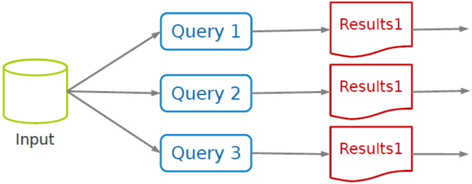


Spark vs. Hadoop

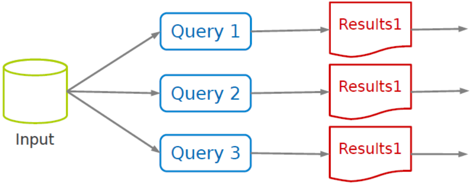


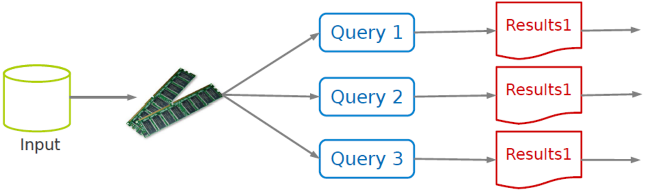


Spark vs. Hadoop



Spark vs. Hadoop





Resilient Distributed Datasets (RDD) (1/2)

- A distributed memory abstraction.

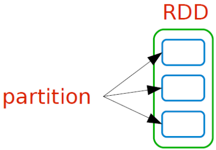
Resilient Distributed Datasets (RDD) (1/2)

- A distributed memory abstraction.

- Immutable collections of objects spread across a cluster.

Resilient Distributed Datasets (RDD) (2/2)

- An RDD is divided into a number of partitions, which are atomic pieces of information.



- Partitions of an RDD can be stored on different nodes of a cluster.

RDD Operators

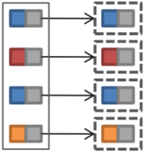
- Higher-order functions: transformations and actions.
- Transformations: lazy operators that create new RDDs.
- Actions: launch a computation and return a value to the program or write data to the external storage.

Transformations vs. Actions

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	RDD[T] $\Rightarrow$ RDD[U]
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	RDD[T] $\Rightarrow$ RDD[T]
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	RDD[T] $\Rightarrow$ RDD[U]
	$sample(fraction : \mathrm{Float})$	:	RDD[T] $\Rightarrow$ RDD[T] (Deterministic sampling)
	$groupByKey()$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, Seq[V])]
	$reduceByKey(f : (\mathrm{V},\mathrm{V}) \Rightarrow \mathrm{V})$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
	$union()$	:	(RDD[T], RDD[T]) $\Rightarrow$ RDD[T]
	$join()$	:	(RDD[(K, V)], RDD[(K, W)]) $\Rightarrow$ RDD[(K, (V, W))]
	$cogroup()$	:	(RDD[(K, V)], RDD[(K, W)]) $\Rightarrow$ RDD[(K, (Seq[V], Seq[W]))]
	$crossProduct()$	:	(RDD[T], RDD[U]) $\Rightarrow$ RDD[(T, U)]
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, W)] (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
Actions	$count()$	:	RDD[T] $\Rightarrow$ Long
	$collect()$	:	RDD[T] $\Rightarrow$ Seq[T]
	$reduce(f : (\mathrm{T},\mathrm{T}) \Rightarrow \mathrm{T})$	:	RDD[T] $\Rightarrow$ T
	$lookup(k : \mathrm{K})$	:	RDD[(K, V)] $\Rightarrow$ Seq[V] (On hash/range partitioned RDDs)
	$save(path : \mathrm{String})$	:	Outputs RDD to a storage system, *e.g.,* HDFS

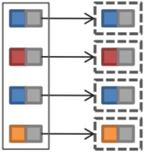
RDD Transformations - Map

- All pairs are independently processed.



RDD Transformations - Map

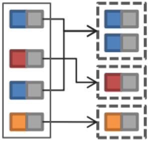
- All pairs are independently processed.



```
// passing each element through a function.
val nums = sc.parallelize(Array(1, 2, 3))
val squares = nums.map(x => x * x) // {1, 4, 9}
```

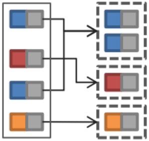
RDD Transformations - GroupBy

- Pairs with identical key are grouped.
- Groups are independently processed.



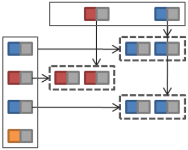
RDD Transformations - GroupBy

- Pairs with identical key are grouped.
- Groups are independently processed.



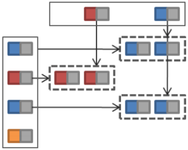
```
val schools = sc.parallelize(Seq(("sics", 1), ("kth", 1), ("sics", 2)))

schools.groupByKey()
// {("sics", (1, 2)), ("kth", (1))}

schools.reduceByKey((x, y) => x + y)
// {("sics", 3), ("kth", 1)}
```

- Performs an equi-join on the key.
- Join candidates are independently processed.



RDD Transformations - Join

- Performs an equi-join on the key.
- Join candidates are independently processed.



```
val list1 = sc.parallelize(Seq(("sics", "10"),
                              ("kth", "50"),
                              ("sics", "20")))

val list2 = sc.parallelize(Seq(("sics", "upsala"),
                              ("kth", "stockholm")))

list1.join(list2)
  // ("sics", ("10", "upsala"))
  // ("sics", ("20", "upsala"))
  // ("kth", ("50", "stockholm"))
```

Basic RDD Actions

- Return all the elements of the RDD as an array.

```
val nums = sc.parallelize(Array(1, 2, 3))
nums.collect() // Array(1, 2, 3)
```

Basic RDD Actions

- Return all the elements of the RDD as an array.

```
val nums = sc.parallelize(Array(1, 2, 3))
nums.collect() // Array(1, 2, 3)
```

- Return an array with the first n elements of the RDD.

```
nums.take(2) // Array(1, 2)
```

Basic RDD Actions

- Return all the elements of the RDD as an array.

```
val nums = sc.parallelize(Array(1, 2, 3))
nums.collect() // Array(1, 2, 3)
```

- Return an array with the first n elements of the RDD.

```
nums.take(2) // Array(1, 2)
```

- Return the number of elements in the RDD.

```
nums.count() // 3
```

Basic RDD Actions

- Return all the elements of the RDD as an array.

```
val nums = sc.parallelize(Array(1, 2, 3))
nums.collect() // Array(1, 2, 3)
```

- Return an array with the first n elements of the RDD.

```
nums.take(2) // Array(1, 2)
```

- Return the number of elements in the RDD.

```
nums.count() // 3
```

- Aggregate the elements of the RDD using the given function.

```
nums.reduce((x, y) => x + y) // 6
```

Creating RDDs

- Turn a collection into an RDD.

```
val a = sc.parallelize(Array(1, 2, 3))
```

- Load text file from local FS, HDFS, or S3.

```
val a = sc.textFile("file.txt")
val b = sc.textFile("directory/*.txt")
val c = sc.textFile("hdfs://namenode:9000/path/file")
```

SparkContext

- Main entry point to Spark functionality.
- Available in shell as variable sc.
- In standalone programs, you should make your own.

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

val sc = new SparkContext(master, appName, [sparkHome], [jars])
```

Spark Hands-on Exercises (1/3)

- Read data from a text file and create an RDD named `pagecounts`.

Spark Hands-on Exercises (1/3)

- Read data from a text file and create an RDD named `pagecounts`.

```
val pagecounts = sc.textFile("hamlet")
```

Spark Hands-on Exercises (1/3)

- Read data from a text file and create an RDD named `pagecounts`.

```
val pagecounts = sc.textFile("hamlet")
```

- Get the first 10 lines of the text file.

Spark Hands-on Exercises (1/3)

- Read data from a text file and create an RDD named `pagecounts`.

```
val pagecounts = sc.textFile("hamlet")
```

- Get the first 10 lines of the text file.

```
pagecounts.take(10).foreach(println)
```

Spark Hands-on Exercises (1/3)

- Read data from a text file and create an RDD named `pagecounts`.

```
val pagecounts = sc.textFile("hamlet")
```

- Get the first 10 lines of the text file.

```
pagecounts.take(10).foreach(println)
```

- Count the total records in the data set `pagecounts`.

Spark Hands-on Exercises (1/3)

- Read data from a text file and create an RDD named `pagecounts`.

```
val pagecounts = sc.textFile("hamlet")
```

- Get the first 10 lines of the text file.

```
pagecounts.take(10).foreach(println)
```

- Count the total records in the data set `pagecounts`.

```
pagecounts.count
```

Spark Hands-on Exercises (2/3)

- Filter the data set `pagecounts` and return the items that have the word `this`, and cache in the memory.

Spark Hands-on Exercises (2/3)

- Filter the data set `pagecounts` and return the items that have the word `this`, and cache in the memory.

```
val linesWithThis = pagecounts.filter(line => line.contains("this")).cache
\\ or
val linesWithThis = pagecounts.filter(_.contains("this")).cache
```

Spark Hands-on Exercises (2/3)

- Filter the data set `pagecounts` and return the items that have the word `this`, and cache in the memory.

```
val linesWithThis = pagecounts.filter(line => line.contains("this")).cache
\\ or
val linesWithThis = pagecounts.filter(_.contains("this")).cache
```

- Find the lines with the most number of words.

Spark Hands-on Exercises (2/3)

- Filter the data set pagecounts and return the items that have the word this, and cache in the memory.

```
val linesWithThis = pagecounts.filter(line => line.contains("this")).cache
\\ or
val linesWithThis = pagecounts.filter(_.contains("this")).cache
```

- Find the lines with the most number of words.

```
linesWithThis.map(line => line.split(" ").size)
  .reduce((a, b) => if (a > b) a else b)
```

Spark Hands-on Exercises (2/3)

- Filter the data set `pagecounts` and return the items that have the word `this`, and cache in the memory.

```
val linesWithThis = pagecounts.filter(line => line.contains("this")).cache
\\ or
val linesWithThis = pagecounts.filter(_.contains("this")).cache
```

- Find the lines with the most number of words.

```
linesWithThis.map(line => line.split(" ").size)
  .reduce((a, b) => if (a > b) a else b)
```

- Count the total number of words.

Spark Hands-on Exercises (2/3)

- Filter the data set `pagecounts` and return the items that have the word `this`, and cache in the memory.

```
val linesWithThis = pagecounts.filter(line => line.contains("this")).cache
\\ or
val linesWithThis = pagecounts.filter(_.contains("this")).cache
```

- Find the lines with the most number of words.

```
linesWithThis.map(line => line.split(" ").size)
  .reduce((a, b) => if (a > b) a else b)
```

- Count the total number of words.

```
val wordCounts = linesWithThis.flatMap(line => line.split(" ")).count
\\ or
val wordCounts = linesWithThis.flatMap(_.split(" ")).count
```

Spark Hands-on Exercises (3/3)

- Count the number of distinct words.

Spark Hands-on Exercises (3/3)

- Count the number of distinct words.

```
val uniqueWordCounts = linesWithThis.flatMap(_.split(" ")).distinct.count
```

Spark Hands-on Exercises (3/3)

- Count the number of distinct words.

```
val uniqueWordCounts = linesWithThis.flatMap(_.split(" ")).distinct.count
```

- Count the number of each word.

Spark Hands-on Exercises (3/3)

- Count the number of distinct words.

```
val uniqueWordCounts = linesWithThis.flatMap(_.split(" ")).distinct.count
```

- Count the number of each word.

```
val eachWordCounts = linesWithThis.flatMap(_.split(" "))
  .map(word => (word, 1))
  .reduceByKey((a, b) => a + b)
```

Spark Streaming

Motivation

- Many applications must process large streams of live data and provide results in real-time.

- Processing information as it flows, without storing them persistently.

Motivation

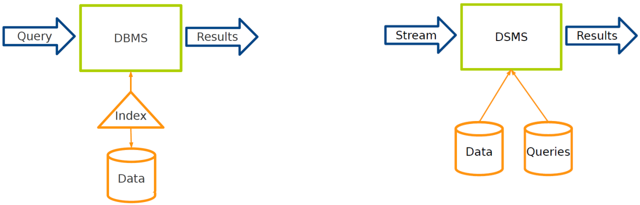
- Many applications must process large streams of live data and provide results in real-time.

- Processing information as it flows, without storing them persistently.

- Traditional DBMSs:
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users.
  - Both aspects contrast with our requirements.

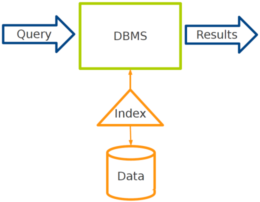
DBMS vs. DSMS (1/3)

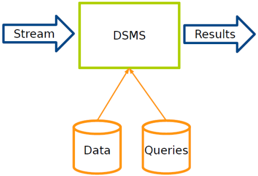
- DBMS: persistent data where updates are relatively infrequent.
- DSMS: transient data that is continuously updated.



DBMS vs. DSMS (2/3)

- DBMS: runs queries just once to return a complete answer.
- DSMS: executes standing queries, which run continuously and provide updated answers as new data arrives.





DBMS vs. DSMS (3/3)

- Despite these differences, DSMSs resemble DBMSs: both process incoming data through a sequence of transformations based on SQL operators, e.g., selections, aggregates, joins.

Spark Streaming

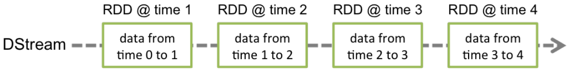
- Run a streaming computation as a series of very small, deterministic batch jobs.

Spark Streaming

- Run a streaming computation as a series of very small, deterministic batch jobs.
  - Chop up the live stream into batches of X seconds.
  - Spark treats each batch of data as RDDs and processes them using RDD operations.
  - Finally, the processed results of the RDD operations are returned in batches.

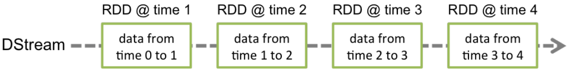
DStream

- DStream: sequence of RDDs representing a stream of data.
  - TCP sockets, Twitter, HDFS, Kafka, ...



DStream

- DStream: sequence of RDDs representing a stream of data.
  - TCP sockets, Twitter, HDFS, Kafka, ...

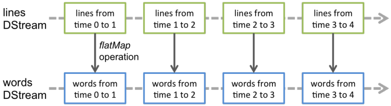


- Initializing Spark streaming

```
val scc = new StreamingContext(master, appName, batchDuration,
[sparkHome], [jars])
```

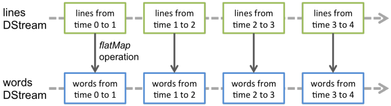
DStream Operations (1/2)

- Transformations: modify data from on DStream to a new DStream.
  - Standard RDD operations (stateless/stateful operations): map, join, ...

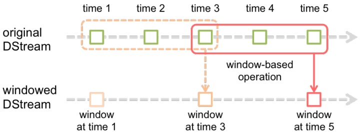


DStream Operations (1/2)

- Transformations: modify data from on DStream to a new DStream.
  - Standard RDD operations (stateless/stateful operations): map, join, ...



  - Window operations: group all the records from a sliding window of the past time intervals into one RDD: window, reduceByAndWindow, ...



Window length: the duration of the window.
Slide interval: the interval at which the operation is performed.

DStream Operations (2/2)

- Output operations: send data to external entity
  - saveAsHadoopFiles, foreach, print, ...

DStream Operations (2/2)

- Output operations: send data to external entity
  - saveAsHadoopFiles, foreach, print, ...

- Attaching input sources

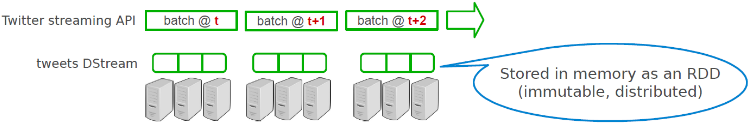
```
ssc.textFileStream(directory)
ssc.socketStream(hostname, port)
```

Example 1 (1/3)

- Get hash-tags from Twitter.

```
val ssc = new StreamingContext("local[2]", "Tweets", Seconds(1))
val tweets = TwitterUtils.createStream(ssc, None)
```

DStream: a sequence of RDD representing a stream of data

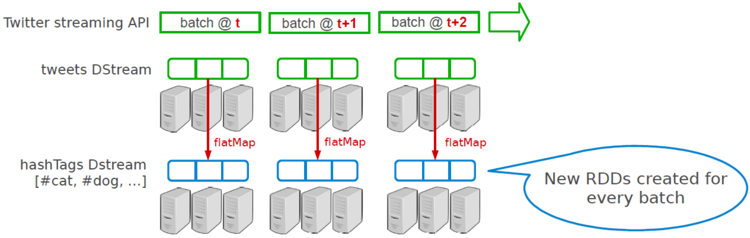


Example 1 (2/3)

- Get hash-tags from Twitter.

```
val ssc = new StreamingContext("local[2]", "Tweets", Seconds(1))
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
```

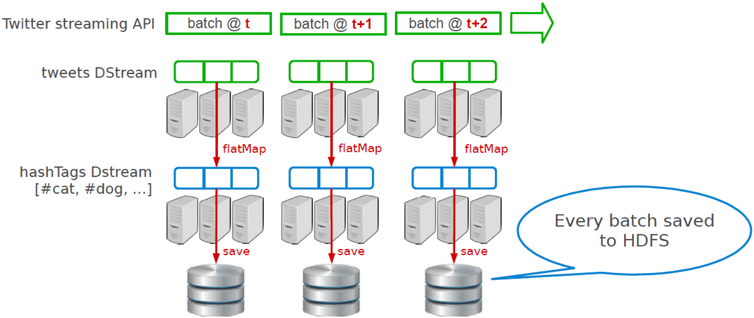
transformation: modify data in one DStream
to create another DStream



Example 1 (3/3)

- Get hash-tags from Twitter.

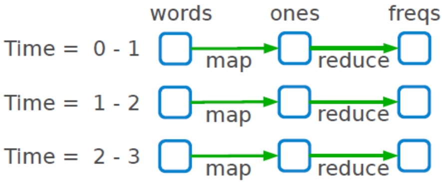
```
val ssc = new StreamingContext("local[2]", "Tweets", Seconds(1))
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```



Example 2

- Count frequency of words received every second.

```
val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
val lines = ssc.socketTextStream(ip, port)
val words = lines.flatMap(_.split(" "))
val ones = words.map(x => (x, 1))
val freqs = ones.reduceByKey(_ + _)
```

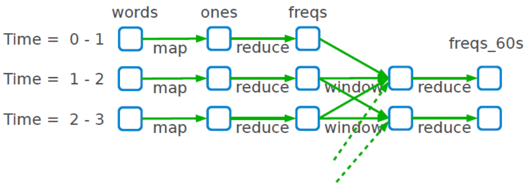


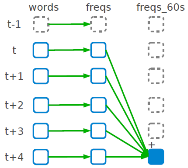
- Count frequency of words received in last minute.

```
val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
val lines = ssc.socketTextStream(ip, port)
val words = lines.flatMap(_.split(" "))
val ones = words.map(x => (x, 1))
val freqs = ones.reduceByKey(_ + _)
val freqs_60s = freqs.window(Seconds(60), Second(1)).reduceByKey(_ + _)
```

window length

window movement





Spark Streaming Hands-on Exercises (1/2)

- Stream data through a TCP connection and port 9999

```
nc -lk 9999
```

Spark Streaming Hands-on Exercises (1/2)

- Stream data through a TCP connection and port 9999

```
nc -lk 9999
```

- import the streaming libraries

Spark Streaming Hands-on Exercises (1/2)

- Stream data through a TCP connection and port 9999

```
nc -lk 9999
```

- import the streaming libraries

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
```

Spark Streaming Hands-on Exercises (1/2)

- Stream data through a TCP connection and port 9999

```
nc -lk 9999
```

- import the streaming libraries

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
```

- Print out the incoming stream every five seconds at port 9999

Spark Streaming Hands-on Exercises (1/2)

- Stream data through a TCP connection and port 9999

```
nc -lk 9999
```

- import the streaming libraries

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
```

- Print out the incoming stream every five seconds at port 9999

```
val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(5))
val lines = ssc.socketTextStream("127.0.0.1", 9999)
lines.print()
```

Spark Streaming Hands-on Exercises (1/2)

- Count the number of each word in the incoming stream every five seconds at port 9999

Spark Streaming Hands-on Exercises (1/2)

- Count the number of each word in the incoming stream every five seconds at port 9999

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._

val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
val lines = ssc.socketTextStream("127.0.0.1", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(x => (x, 1))
val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.print()
```

Spark Streaming Hands-on Exercises (2/2)

- Extend the code to generate word count over last 30 seconds of data, and repeat the computation every 10 seconds

Spark Streaming Hands-on Exercises (2/2)

- Extend the code to generate word count over last 30 seconds of data, and repeat the computation every 10 seconds

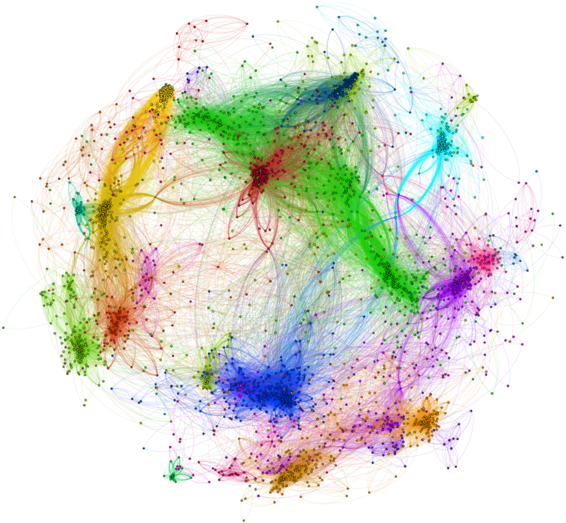
```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._

val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(5))
val lines = ssc.socketTextStream("127.0.0.1", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val windowedWordCounts = pairs
  .reduceByKeyAndWindow(_ + _, _ - _, Seconds(30), Seconds(10))
windowedWordCounts.print()
wordCounts.print()
```

GraphX

Introduction

- Graphs provide a flexible abstraction for describing relationships between discrete objects.
- Many problems can be modeled by graphs and solved with appropriate graph algorithms.

Large Graph

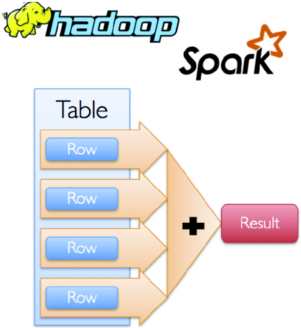


Large-Scale Graph Processing

- Large graphs need large-scale processing.
- A large graph either cannot fit into memory of single computer or it fits with huge cost.

Question

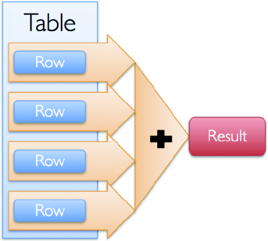
Can we use platforms like MapReduce or Spark, which are based on data-parallel model, for large-scale graph proceeding?



Data-Parallel Model for Large-Scale Graph Processing

- The platforms that have worked well for developing parallel applications are not necessarily effective for large-scale graph problems.

- Why?



Graph Algorithms Characteristics (1/2)

- Unstructured problems
  - Difficult to extract parallelism based on partitioning of the data: the irregular structure of graphs.
  - Limited scalability: unbalanced computational loads resulting from poorly partitioned data.

Graph Algorithms Characteristics (1/2)

- Unstructured problems
  - Difficult to extract parallelism based on partitioning of the data: the irregular structure of graphs.
  - Limited scalability: unbalanced computational loads resulting from poorly partitioned data.

- Data-driven computations
  - Difficult to express parallelism based on partitioning of computation: the structure of computations in the algorithm is not known a priori.
  - The computations are dictated by nodes and links of the graph.

Graph Algorithms Characteristics (2/2)

- Poor data locality
  - The computations and data access patterns do not have much locality: the irregular structure of graphs.

Graph Algorithms Characteristics (2/2)

- Poor data locality
  - The computations and data access patterns do not have much locality: the irregular structure of graphs.

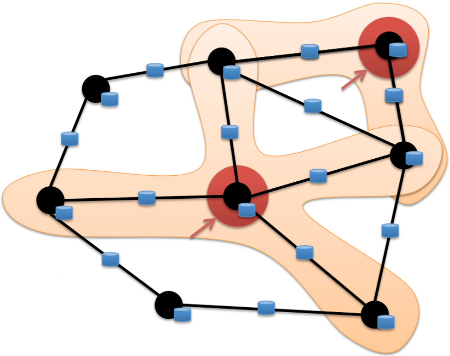
- High data access to computation ratio
  - Graph algorithms are often based on exploring the structure of a graph to perform computations on the graph data.
  - Runtime can be dominated by waiting memory fetches: low locality.

Proposed Solution

Graph-Parallel Processing

Proposed Solution

Graph-Parallel Processing

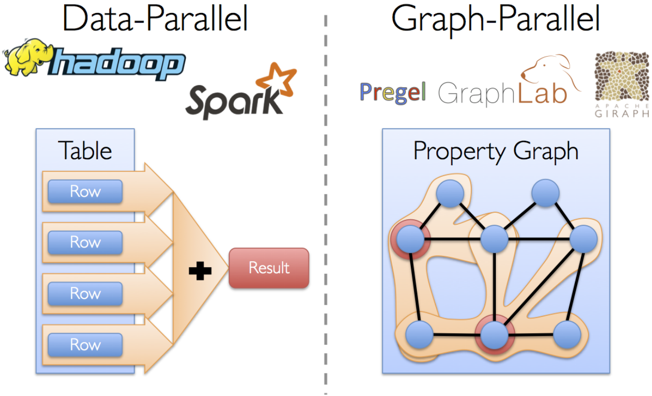


- Computation typically depends on the neighbors.

Graph-Parallel Processing

- Restricts the types of computation.
- New techniques to partition and distribute graphs.
- Exploit graph structure.
- Executes graph algorithms orders-of-magnitude faster than more general data-parallel systems.

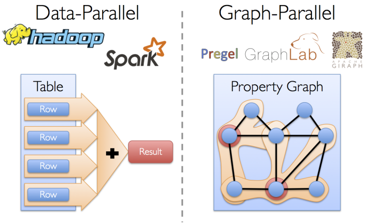
Data-Parallel vs. Graph-Parallel Computation



Data-Parallel vs. Graph-Parallel Computation

- Data-parallel computation
  - Record-centric view of data.
  - Parallelism: processing independent data on separate resources.

- Graph-parallel computation
  - Vertex-centric view of graphs.
  - Parallelism: partitioning graph (dependent) data across processing resources, and resolving dependencies (along edges) through iterative computation and communication.



Graph-Parallel Computation Frameworks

APACHE GIRAPH

GraphLab

Pregel

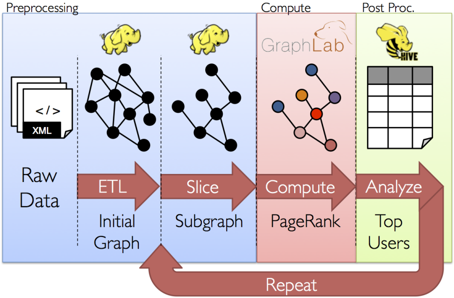
Data-Parallel vs. Graph-Parallel Computation

- Graph-parallel computation: restricting the types of computation to achieve performance.

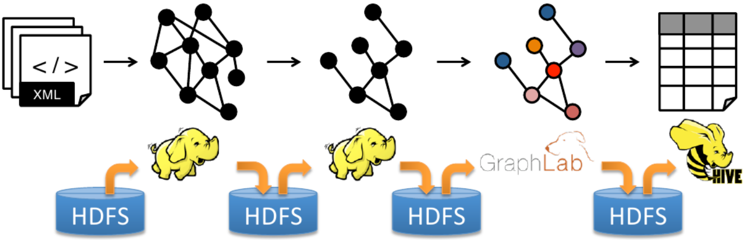
Data-Parallel vs. Graph-Parallel Computation

- Graph-parallel computation: restricting the types of computation to achieve performance.

- But, the same restrictions make it difficult and inefficient to express many stages in a typical graph-analytics pipeline.

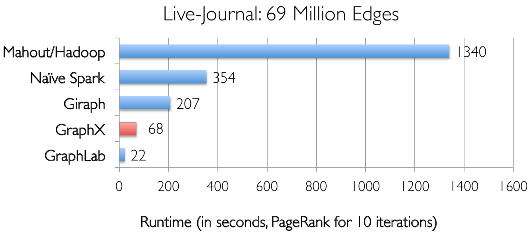


Data-Parallel and Graph-Parallel Pipeline

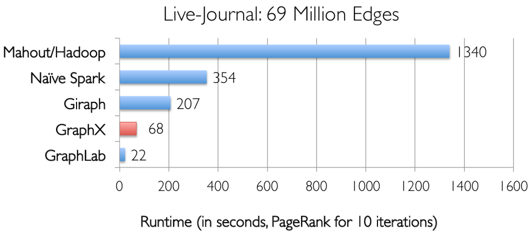


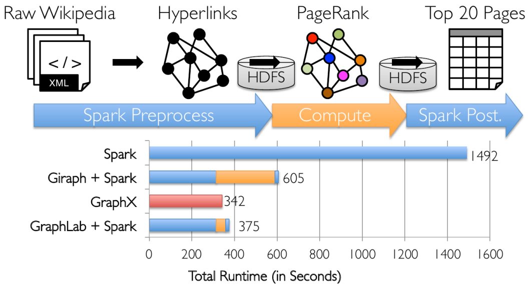
- Moving between table and graph views of the same physical data.
- Inefficient: extensive data movement and duplication across the network and file system.

GraphX vs. Data-Parallel/Graph-Parallel Systems



GraphX vs. Data-Parallel/Graph-Parallel Systems



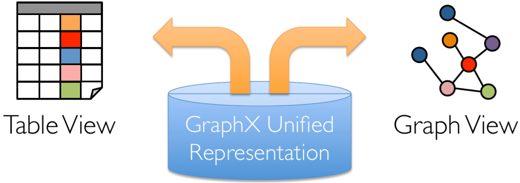


GraphX

- New API that blurs the distinction between Tables and Graphs.
- New system that unifies Data-Parallel and Graph-Parallel systems.
- It is implemented on top of Spark.

Unifying Data-Parallel and Graph-Parallel Analytics

- Tables and Graphs are composable views of the same physical data.
- Each view has its own operators that exploit the semantics of the view to achieve efficient execution.

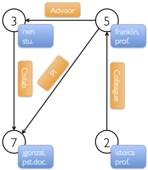


Data Model

- Property Graph: represented using two Spark RDDs:
  - Edge collection: VertexRDD
  - Vertex collection: EdgeRDD

```scala
// VD: the type of the vertex attribute
// ED: the type of the edge attribute
class Graph[VD, ED] {
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED, VD]
}
```

Property Graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

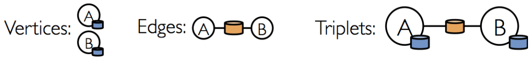
Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Primitive Data Types

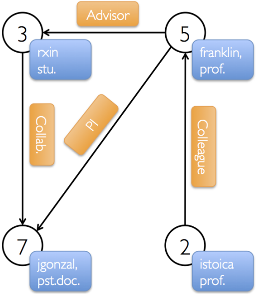
```
// Vertex collection
class VertexRDD[VD] extends RDD[(VertexId, VD)]

// Edge collection
class EdgeRDD[ED] extends RDD[Edge[ED]]
case class Edge[ED, VD](srcId: VertexId = 0, dstId: VertexId = 0,
                    attr: ED = null.asInstanceOf[ED])

// Edge Triple
class EdgeTriplet[VD, ED] extends Edge[ED]
```

- EdgeTriplet represents an edge along with the vertex attributes of its neighboring vertices.



Example (1/3)

Property Graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Example (2/3)

```
val sc: SparkContext

// Create an RDD for the vertices
val users: RDD[(Long, (String, String))] = sc.parallelize(
    Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
          (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))

// Create an RDD for edges
val relationships: RDD[Edge[String]] = sc.parallelize(
    Array(Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
          Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))

// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")

// Build the initial Graph
val userGraph: Graph[(String, String), String] =
    Graph(users, relationships, defaultUser)
```

Example (3/3)

```
// Constructed from above
val userGraph: Graph[(String, String), String]

// Count all users which are postdocs
userGraph.vertices.filter { case (id, (name, pos)) => pos == "postdoc" }.count

// Count all the edges where src > dst
userGraph.edges.filter(e => e.srcId > e.dstId).count

// Use the triplets view to create an RDD of facts
val facts: RDD[String] = graph.triplets.map(triplet =>
    triplet.srcAttr._1 + " is the " +
    triplet.attr + " of " + triplet.dstAttr._1)

// Remove missing vertices as well as the edges to connected to them
val validGraph = graph.subgraph(vpred = (id, attr) => attr._2 != "Missing")

facts.collect.foreach(println)
```

Property Operators (1/2)

```
class Graph[VD, ED] {
  def mapVertices[VD2](map: (VertexId, VD) => VD2): Graph[VD2, ED]

  def mapEdges[ED2](map: Edge[ED] => ED2): Graph[VD, ED2]

  def mapTriplets[ED2](map: EdgeTriplet[VD, ED] => ED2): Graph[VD, ED2]
}
```

- They yield new graphs with the vertex or edge properties modified by the map function.
- The graph structure is unaffected.

Property Operators (2/2)

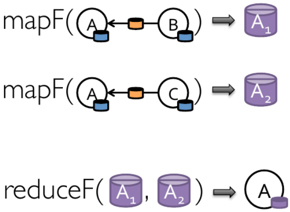
```
val newGraph = graph.mapVertices((id, attr) => mapUdf(id, attr))
```

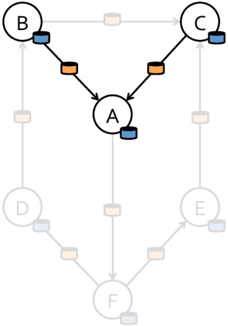
```
val newVertices = graph.vertices.map((id, attr) => (id, mapUdf(id, attr)))
val newGraph = Graph(newVertices, graph.edges)
```

- Both are logically equivalent, but the second one does not preserve the structural indices and would not benefit from the GraphX system optimizations.

Map Reduce Triplets

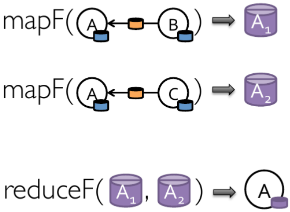
- Map-Reduce for each vertex

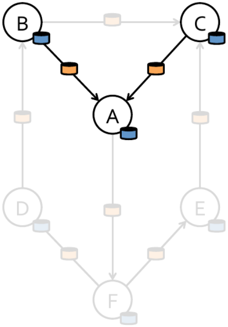




Map Reduce Triplets

- Map-Reduce for each vertex






```
// what is the age of the oldest follower for each user?
val oldestFollowerAge = graph.mapReduceTriplets(
  e => Iterator((e.dstAttr, e.srcAttr)), // Map
  (a, b) => max(a, b) // Reduce
).vertices
```

Structural Operators

```
class Graph[VD, ED] {
  // returns a new graph with all the edge directions reversed
  def reverse: Graph[VD, ED]

  // returns the graph containing only the vertices and edges that satisfy
  // the vertex predicate
  def subgraph(epred: EdgeTriplet[VD,ED] => Boolean,
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]

  // a subgraph by returning a graph that contains the vertices and edges
  // that are also found in the input graph
  def mask[VD2, ED2](other: Graph[VD2, ED2]): Graph[VD, ED]
}
```

Structural Operators Example

```scala
// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)

// Run Connected Components
val ccGraph = graph.connectedComponents()

// Remove missing vertices as well as the edges to connected to them
val validGraph = graph.subgraph(vpred = (id, attr) => attr._2 != "Missing")

// Restrict the answer to the valid subgraph
val validCCGraph = ccGraph.mask(validGraph)
```

Questions?