Big Data Technologies

Julián Arenas Guerrero
julian.arenas.guerrero@upm.es

With some slides from Jesús Montes

Nov. 2024

Big Data technologies

Big Data problems require new tools and techniques. These can be grouped in three categories:

1. **Infrastructure**
    - Collecting/storing the data
    - Processing the data
2. **Analytics**
    - Extracting knowledge from data
    - Visualizing the data/knowledge
3. Applications

In the last 15 years, the most popular software has been probably **Hadoop**.

Hadoop

- The first implementation of MapReduce (by Google) was proprietary.
- Hadoop was developed as an Open Source version of MapReduce.
- Hadoop included its own implementation of the MapReduce engine, and also of the Google File System (GFS), called Hadoop File System (HDFS).
- Implemented in Java.
- Highly scalable and extensible.
- In time, Hadoop has grown from a MapReduce implementation to a full cluster management framework.
- The *de facto* standard for Big Data clusters.

Hadoop

The Hadoop framework is designed to be installed in a standard computing cluster. Hadoop is divided in two separated main components:

1. Hadoop File System (HDFS)
    - Organizes data into files, and distributes them through the cluster.
    - It has fault tolerance and high availability capabilities.
2. Application manager
    - Manages the applications being executed in the cluster.
    - It is called YARN (Yet Another Resource Negotiator) and is now a general purpose resource manager.

Hadoop File System (HDFS)

HDFS organizes data files in the cluster, so they can be accessed by any application running on top of YARN.

HDFS has a master/slave architecture and it is composed of two main services:

- **NameNode**: The central component, containing files metadata. It can be replicated (for high availability and fault tolerance) and configured in a primary-backup setup (using a SecondaryNameNode).
- **DataNode**: It runs in each cluster node that stores data. It stores data in the node's local file system.

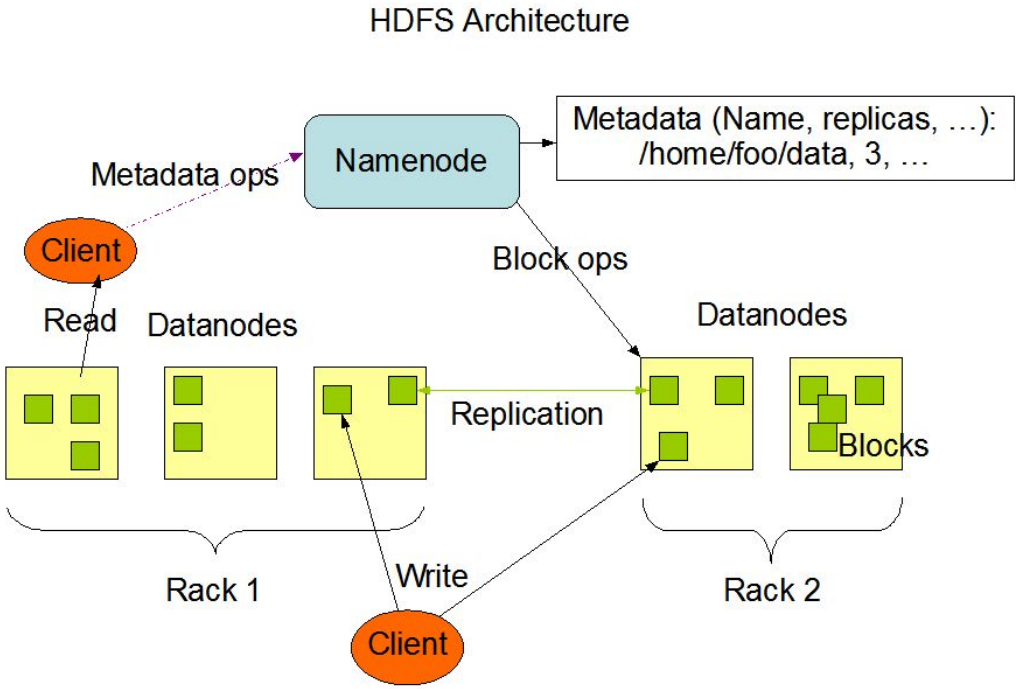
Hadoop File System (HDFS)

Assumptions and goals:

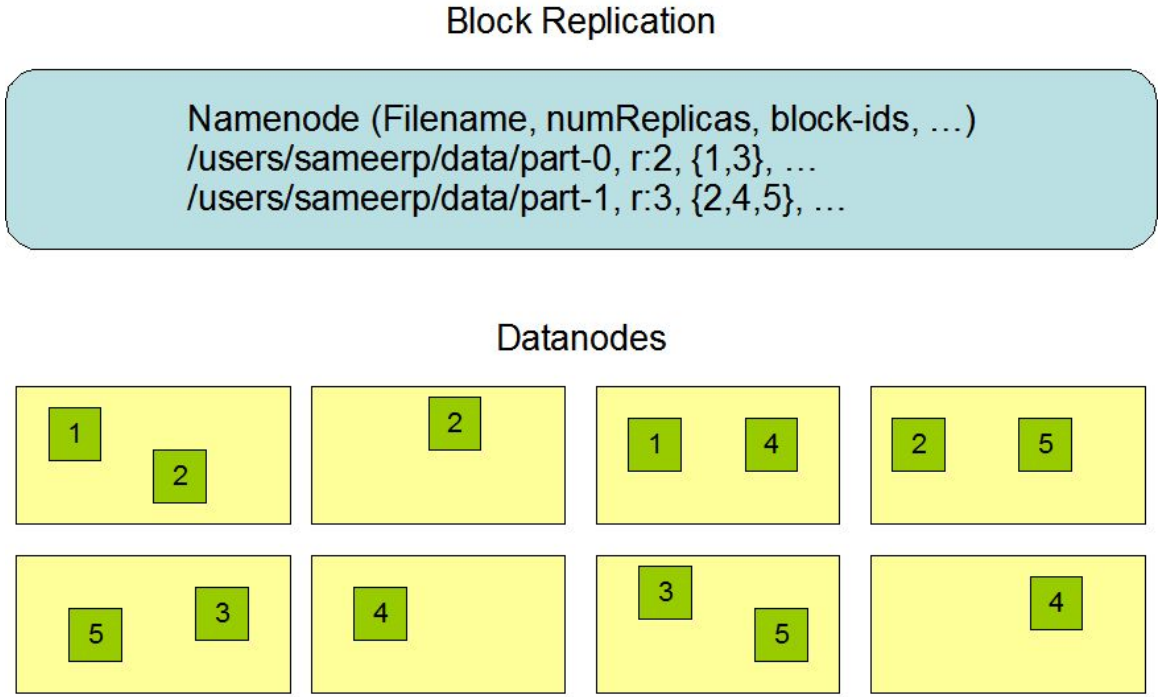
- (Hardware) failures are the norm rather than the exception.
- Emphasis on high throughput of data access rather than low latency of data access.
- Typical applications that run on HDFS have large data sets. HDFS is tuned to support large files.
- Most files are mutated by appending new data rather than overwriting existing data. Write once read many.
- “Moving computation is cheaper than moving data”

Hadoop File System (HDFS)

- HDFS splits files into blocks and distributes these blocks throughout the cluster.
- Default block size is 128 MB .
- DataNodes store blocks in a dedicated directory of the local file system (with subdirectories).
- Block replication provides fault tolerance.
- HDFS is designed to store very large files, and it is not efficient with small ones.
- User data never flows through the NameNode



Hadoop File System (HDFS)



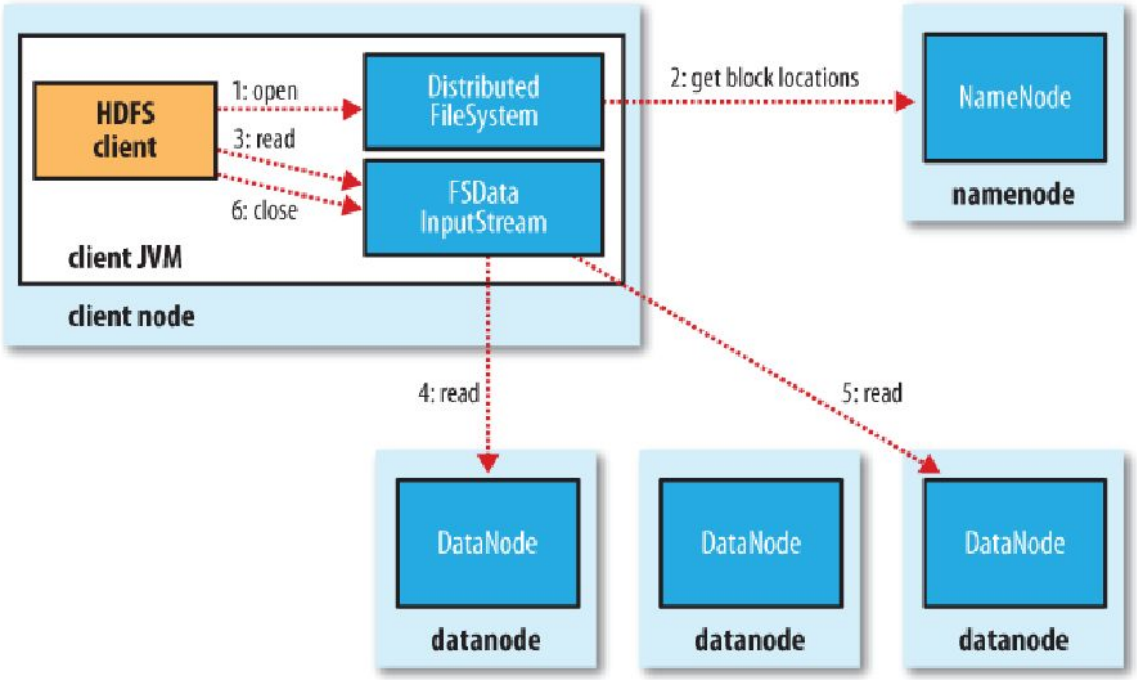
Hadoop File System (HDFS)

- HDFS files are not part of the local file system (although its blocks are stored on it).
- To access HDFS, we need to use its own API.
  - Hadoop comes with a command-line tool that allows us to perform basic operations in HDFS.
  - This command-line tool is implemented using the HDFS API.
- To access a file, a client needs to contact the NameNode first, to get the file blocks locations. Afterwards, blocks can be read in parallel.
- In a basic HDFS configuration, the NameNode becomes a single point of failure. Therefore its metadata needs to be backed-up regularly, to avoid information loss.

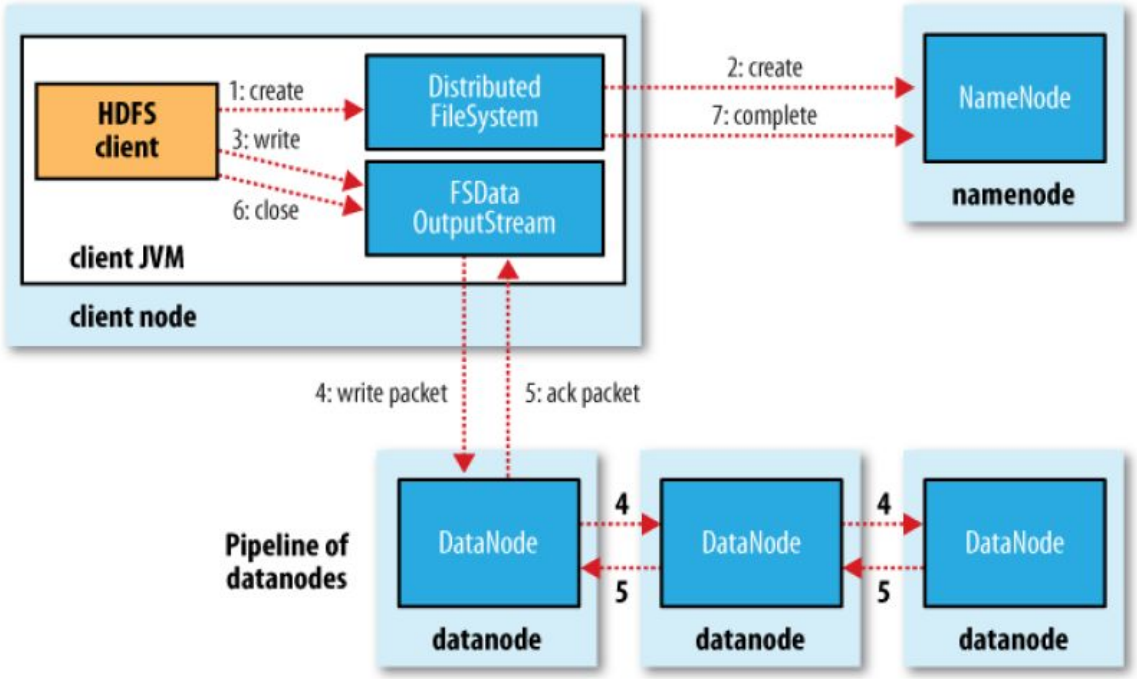
Hadoop File System (HDFS)

- Replica placement follows a rack-aware policy.
- Replica selection: that closest to the reader.
- Default replication factor: 3.
- NameNode periodically receives a Heartbeat from DataNodes and a blockreport. If Heartbeat is not received for 10 minutes, DataNode is considered dead.

Reading a file in HDFS



Writing a file in HDFS



HDFS command line tool

- In a machine with Hadoop installed, the command *hadoop* is the standard command-line interface for Hadoop.
- Running *hadoop fs* will show all command-line operations for HDFS.
- The program includes tools to:
  - Print files (cat)
  - Add local files to HDFS (put)
  - Get files from HDFS to the local file system (get)
  - Remove files (rm)
  - Rename/move files (mv)
  - ...

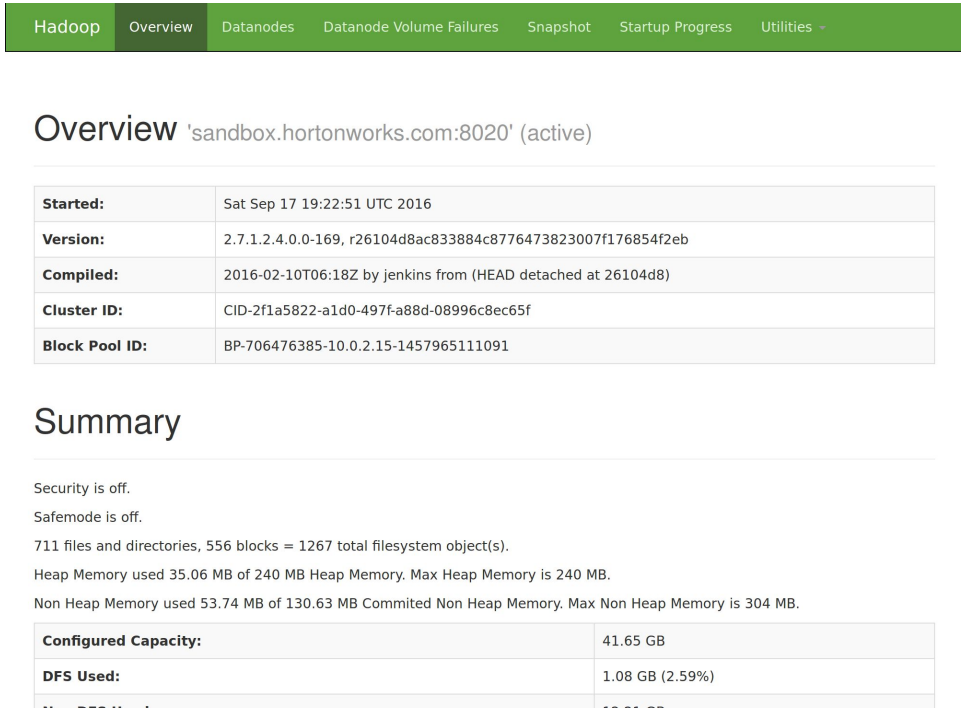
```
# hadoop fs
Usage: hadoop fs [generic options]
    [-appendToFile <localsrc> ... <dst>]
    [-cat [-ignoreCrc] <src> ...]
    [-checksum <src> ...]
    [-chgrp [-R] GROUP PATH...]
    [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
    [-chown [-R] [OWNER][:[GROUP]] PATH...]
    [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
    [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ...
<localdst>]
    [-count [-q] [-h] [-v] [-t [<storage type>]] <path> ...]
    [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
    [-createSnapshot <snapshotDir> [<snapshotName>]]
    [-deleteSnapshot <snapshotDir> <snapshotName>]
    [-df [-h] [<path> ...]]
    [-du [-s] [-h] <path> ...]
    [-expunge]
    [-find <path> ... <expression> ...]
    [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
    [-getfacl [-R] <path>]
    [-getfattr [-R] {-n name | -d} [-e en] <path>]
    [-getmerge [-nl] <src> <localdst>]
    [-help [cmd ...]]
    [-ls [-d] [-h] [-R] [<path> ...]]
```

. . .

NameNode web UI

The HDFS NameNode also provides a web UI (by default on port 50070).

- Status information
- Monitoring
- Operation logs
- Basic file browsing



YARN

YARN (Yet Another Resource Negotiator) is Hadoop's general purpose application manager. The data-computation framework is formed by:

- **ResourceManager**: The central component. It is the ultimate authority that arbitrates resources among all the applications in the system.
- **NodeManager**: It runs in each cluster node that performs computation. It executes and monitors tasks and reporting to the ResourceManager.

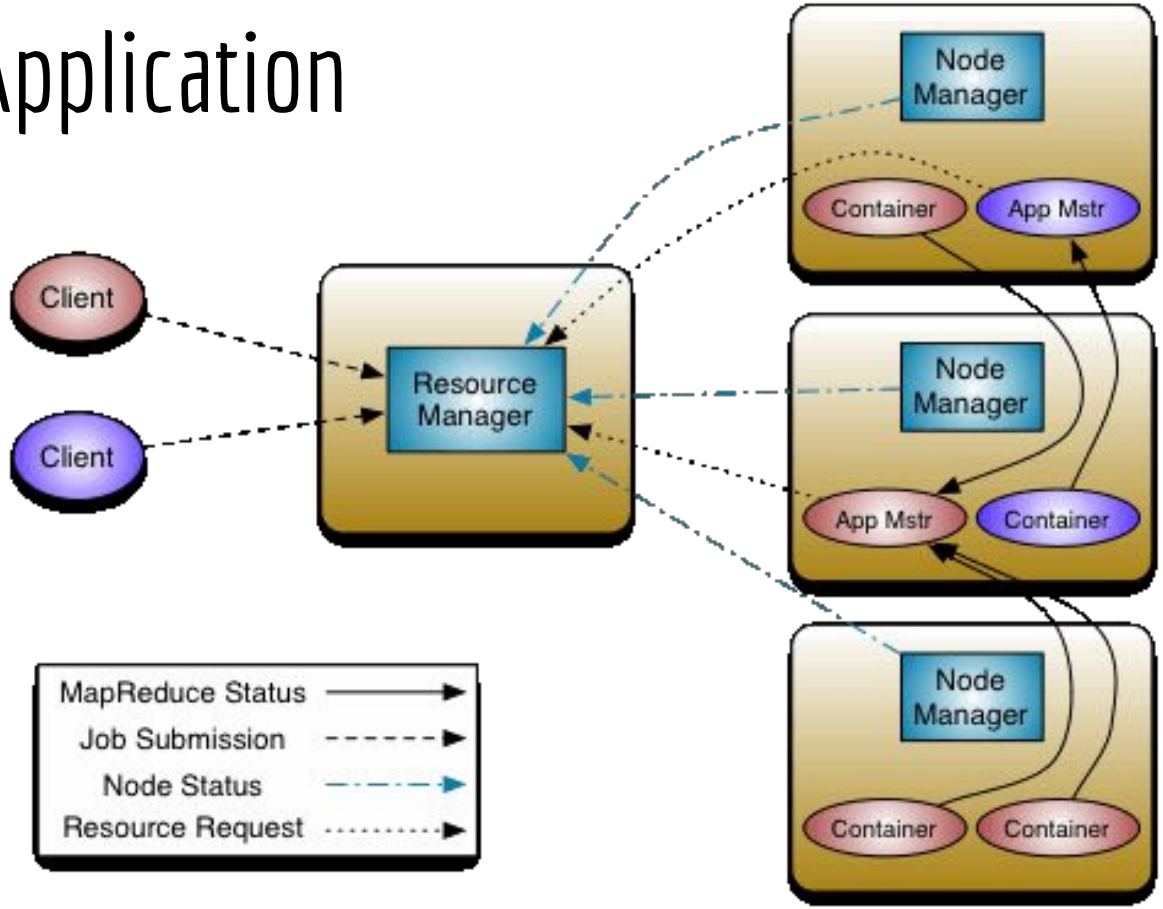
The per-application *ApplicationMaster* is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

YARN

The ResourceManager contains two parts with distinct functions:

- The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.
- The *Scheduler* is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application.

YARN Application



YARN

- YARN incorporates several fault-tolerance and high availability features:
  - Redundant ResourceManagers.
  - Task failure detection and relaunch.
- The YARN Scheduler has a pluggable policy plug-in. It currently includes two policy implementations (CapacityScheduler and FairScheduler), but more can be added.
- YARN supports resource allocation restrictions, such as users quotas, executions queues, etc.
- YARM federation: wires together multiple yarn (sub-)clusters, and make them appear as a single massive cluster.

YARN command line tool

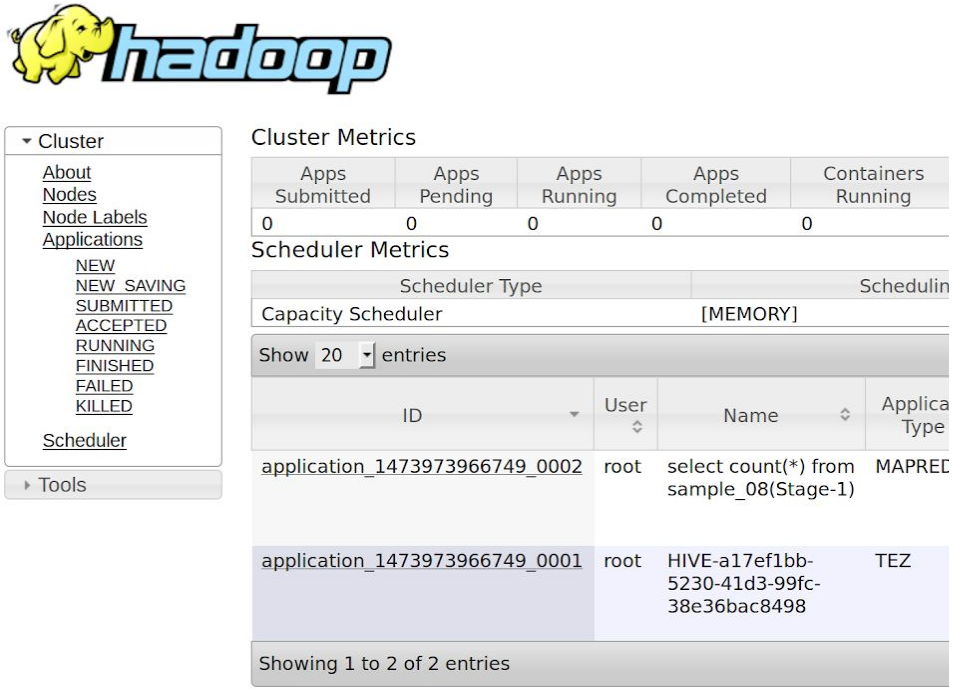
- In a machine with Hadoop installed, the command *yarn* is the standard command-line interface for YARN.
- Running *yarn* will show all command-line operations available.
- The *yarn* command can be used to submit applications to the cluster.
- If the application is packaged in a JAR file, *yarn jar* can be used.
- *yarn application* can be used to monitor the status of applications being executed.

```
# yarn application
[...]
usage: application
 -appStates <States>          Works with -list to filter applications
                              based on input comma-separated list of
                              application states. The valid application
                              state can be one of the following:
                              ALL,NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUN
                              NING,FINISHED,FAILED,KILLED
 -appTypes <Types>            Works with -list to filter applications
                              based on input comma-separated list of
                              application types.
 -help                        Displays help for all commands.
 -kill <Application ID>       Kills the application.
 -list                        List applications. Supports optional use
                              of -appTypes to filter applications based
                              on application type, and -appStates to
                              filter applications based on application
                              state.
 -movetoqueue <Application ID>   Moves the application to a different
                              queue.
 -queue <Queue Name>          Works with the movetoqueue command to
                              specify which queue to move an
                              application to.
 -status <Application ID>     Prints the status of the application.
```

ResourceManager web UI

The Yarn ResourceManager also provides a web UI (by default on port 8088).

- System status information
- Application monitoring
- Scheduler information
- Operation logs



First steps with Hadoop

As a first example, we are going to run WordCount on a text file using Hadoop in pseudo-distributed mode:

1. Start NameNode daemon and DataNode daemon.
   ```
   $ sbin/start-dfs.sh
   ```
2. Make the HDFS directories required to execute MapReduce jobs:
   ```
   $ bin/hdfs dfs -mkdir -p /user/<username>
   ```
3. Download the book “A Tale of Two Cities” by Charles Dickens from Project Gutenberg:
   ```
   $ wget https://www.gutenberg.org/files/98/98-0.txt
   ```
   (or just use any other txt file you want)

First steps with Hadoop

4. Create a directory in HDFS for our book

   ```
   $ bin/hdfs dfs -mkdir -p tmp/book
   ```

5. Copy the book file to HDFS

   ```
   $ bin/hdfs dfs -put 98-0.txt tmp/book/
   ```

6. Check the book file has been stored in HDFS

   ```
   $ bin/hdfs dfs -ls tmp/book
   ```

7. Run the MapReduce application

   ```
   $ bin/hadoop jar
   share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar
   wordcount tmp/book/98-0.txt tmp/wc_output
   ```

First steps with Hadoop

8. When the job finishes, check the contents of the output directory:

   ```
   $ bin/hdfs dfs -ls tmp/wc_output
   ```

9. Finally, take a look at the results:

   ```
   $ bin/hdfs dfs -cat tmp/wc_output/part-r-00000
   ```

10. When you're done, stop the daemons with:

    ```
    $ sbin/stop-dfs.sh
    ```

For this example, we have used the Word Count implementation provided in the Hadoop MapReduce examples JAR file included in Hadoop.

The Hadoop ecosystem

- Hadoop provides HDFS and MapReduce/YARN.
- Based on these tools, many other Big Data tools have been created.
- All these tools are designed to operate with Hadoop, and are commonly considered as part of the *Hadoop Ecosystem*.
- Most of these technologies are Open Source, Apache projects.
- The complete list of technologies is extremely large.

- **Hive**: A data warehouse implemented on top of HDFS. It provides a query language very similar to SQL. Data operations are implemented in MapReduce.
- **Pig**: High level programming language for ETL (extraction-transformation-load) processes. Implemented in MapReduce.
- **Sqoop**: Tool for connecting Hadoop to a RDBMS.
- **HBase**: Wide-column store
- ...

Hadoop installation and configuration

- Hadoop can be downloaded from its official web page (http://hadoop.apache.org/) and installed/configured following the official documentation.
- This is, however, a complex task, as many configuration aspects have to be taken into account.
  - Software installation in all nodes
  - Cluster configuration
  - Security aspects
  - ...
- When other tools from the Hadoop ecosystem are added, the complexity increases even more.

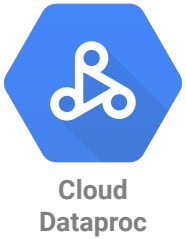
Hadoop as a Service (IaaS + Hadoop)

Nowadays, all major Cloud vendors provide Hadoop services

Typically, a Hadoop cluster can be easily allocated, selecting parameters such as:

- Number of nodes
- Node size/computing power
- Hadoop version
- Additional software included, from the Hadoop ecosystem (Hive, etc.)

Hadoop services are deployed over IaaS resources (VMs, cloud storage, etc.)



Apache Hive

- Hive is a data warehouse system built on top of Hadoop
- It was originally developed by Facebook, but it is now an Apache project.
- Hive allows to organize the data stored into HDFS in schemas (databases) and tables.
- It has its own query language (HiveQL), similar to SQL.
- HiveQL queries are implemented as MapReduce jobs.

Apache Hive

Hive can be accessed via several different interfaces:

- HiveQL console (launched with the *hive* command or the *beeline* client)
- JDBC driver
- Apache Thrift, HCatalog...

Hive can serve as the basis for storing and organizing data in a Big Data project. Its familiar table-like approach helps data management. Hive data is easily accessible by any other application running in the Hadoop cluster.

Apache Hive

Word count with Hive:

```
DROP TABLE IF EXISTS docs;
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'input_file' INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
 (SELECT explode(split(line, '\s')) AS word FROM docs) temp
GROUP BY word
ORDER BY word;
```

Apache Hive

Hive execution plan:

```
EXPLAIN CREATE TABLE flights_by_carrier AS
SELECT carrier, COUNT(flight) AS num
FROM flights
GROUP BY carrier;
```

```
STAGE DEPENDENCIES:
    Stage-1 is a root stage
    Stage-0 depends on stages: Stage-1
    Stage-3 depends on stages: Stage-0
    Stage-2 depends on stages: Stage-3

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan      [...]
      Reduce Operator Tree:
            Group By Operator     [...]
  Stage: Stage-0
      Move Operator
        files:
              hdfs directory: true
              destination: hdfs://localhost:8020/[...]/fly.db/flights_by_carrier
  Stage: Stage-3
    Create Table Operator:
      Create Table
          columns: carrier string, num bigint
          [...]
  Stage: Stage-2
      Stats-Aggr Operator
```

MapReduce

HDFS

MetaStore

Apache Hive

Hive partitioning is a partitioning strategy that is used to split a table into multiple files based on partition keys. The files are organized into folders. Within each folder, the partition key has a value that is determined by the name of the folder.

```
orders
├── year=2021
│   ├── month=1
│   │   ├── file1.parquet
│   │   └── file2.parquet
│   └── month=2
│       └── file3.parquet
└── year=2022
    ├── month=11
    │   ├── file4.parquet
    │   └── file5.parquet
    └── month=12
        └── file6.parquet
```

Apache Hive

Filters on the partition keys can be automatically pushed **down** into the files. This way the system skips reading files that are not necessary to answer a query.

```
SELECT *
FROM orders/*/*/*.parquet
WHERE year = 2022 AND month = 11;
```

```
orders
└── year=2022
    └── month=11
        ├── file4.parquet
        └── file5.parquet
```

Apache Parquet

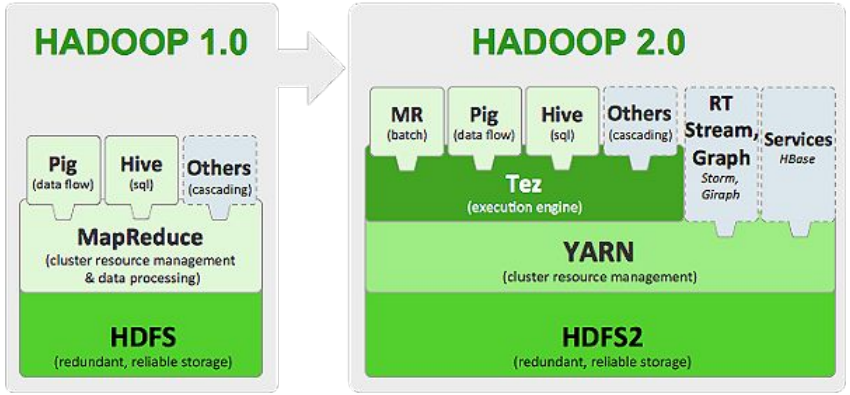
- Columnar data storage format in the Hadoop ecosystem.
- Designed by Cloudera and Twitter.
- Similar to Apache Orc and RCFile, but more popular.
- Scans only projected columns
- Column-wise data compression
  - Run-length encoding (RLE) -> *sorting*
  - Dictionary encoding
  - Delta encoding
  - Bit packing
- Efficient data access with zonemaps/block range index *-> sorting*

Beyond MapReduce

- MapReduce is a ground-breaking technology that provided the basis for many Big Data initiatives.
- MapReduce, however, presents several important limitations, that have motivated the development of newer, more powerful alternatives.
  - Restrictive programming model
  - Heavy dependency on local disk read/write operations
  - Unaware of internal data structure
  - …
- These “MapReduce descendants” try to improve on the MapReduce paradigm, incorporating new features and increasing performance.
- Representative examples: Tez, Spark.

Apache Tez

- Tez generalizes the MapReduce paradigm by treating computations as Directed Acyclic Graphs (DAGs).
- It is based on expressing computations as a dataflow graph.
- Tez is not meant directly for end-users, but as replacement for MapReduce for high level applications.
- Built on top of YARN.
- Tez uses in-memory computation extensively, removing I/O stress and improving performance.
- Plan reconfiguration at runtime and dynamic physical data flow decisions

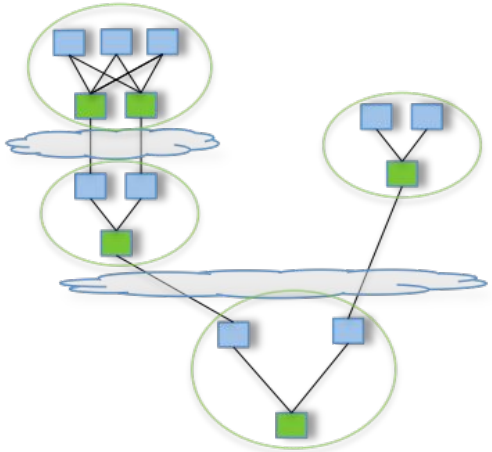


Apache Tez

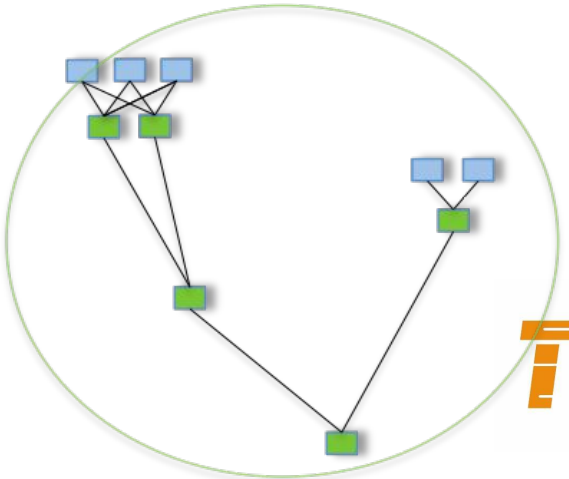
```
1: DAG dag = DAG.create("WordCount");

2: Vertex tokenizerVertex = Vertex.create("Tokenizer", TokenProcessor.class)
                              .addDataSource("Input", HdfsInitializer.class);

3: Vertex summationVertex = Vertex.create("Summation", SumProcessor.class)
                              .addDataSink("Output", HdfsCommitter.class);

4: EdgeProperty edgeProperty = EdgeProperty.create(scatter_gather,
                              KeyValueShuffleWriter.class, KeyValueShuffleReader.class);

5: dag.addVertex(tokenizerVertex).addVertex(summationVertex)
      .addEdge(Edge.create(tokenizerVertex, summationVertex, edgeProperty);
```

Tez vs MapReduce:

- Eliminates job launch overhead of workflow jobs.
- Eliminates extra write/reads in every workflow job.
- Better locality because the engine has the global picture.



Pig/Hive - MR



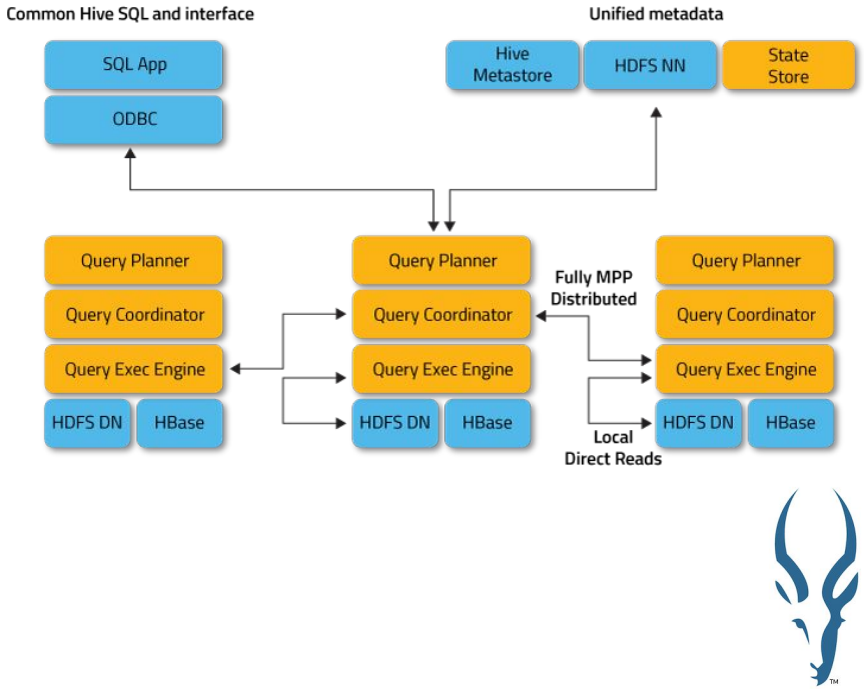
Pig/Hive - Tez

Apache Spark

- Defined by its creators as a “general purpose cluster computing system”
- Developed in Scala. Provides APIs in Scala, Java, Python and R.
- Spark is meant for end-users.
- Centered on a data structure called resilient distributed dataset (RDD)
- Based on principles similar to MapReduce, but allowing a more flexible dataflow structure.
- Like Tez, it uses in-memory computation heavily to optimize performance.
- Nowadays, it is the most popular MapReduce successor.

Apache Impala

- Analytics database built on top of Hadoop, with SQL interface.
- Provides low latency and high concurrency for BI/analytic queries.
- Utilizes Hadoop file formats and Hive's metadata and ODBC driver.
- Instead of MapReduce, implements its own specialized distributed query engine.
- Faster than Hive (with MapReduce), but less fault-tolerant and not suited for time-consuming batch processing.



Apache Kudu

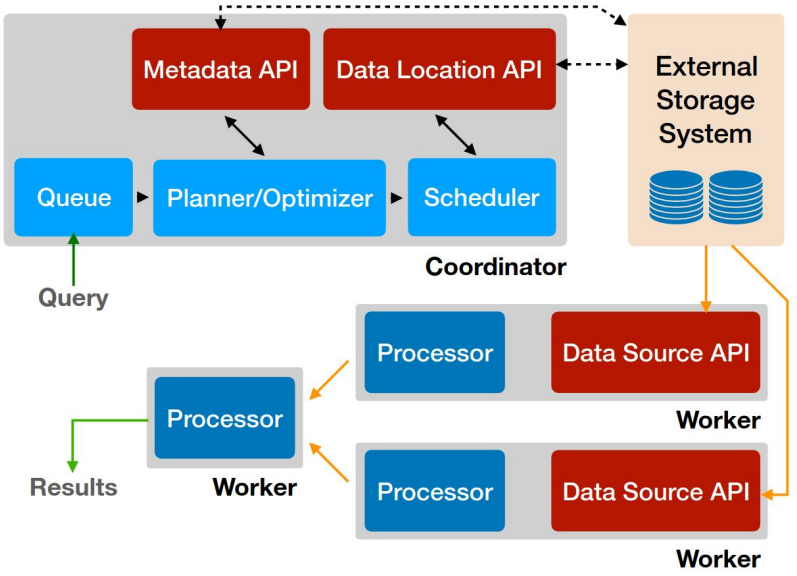
- Column-based data store for Hadoop.
- Fast processing of OLAP workloads.
- Integration with MapReduce, Spark and other Hadoop ecosystem components.
- Custom API (No SQL)
- Tight integration with Apache Impala.
- Strong performance for running sequential and random workloads simultaneously.
- High availability.

Amazon S3

- Simple Storage Service (S3) is a storage service provided by Amazon Web Services (AWS).
- Flat structure (non hierarchical), with data stored as objects in resources known as buckets.
- Metadata tags and prefixes can also be defined to organize data.
- It’s main aim is to provide high scalability, high availability, high consistency with a low access latency.
- Supports data versioning.
- Integrates with other AWS services for querying, archiving, processing, monitoring…

Presto overview

- A coordinator and multiple workers
- **Coordinator** is responsible for admitting, parsing, planning, optimizing and orchestrating queries
- **Workers** nodes are responsible for query processing
- Intermediate data and state is stored **in-memory**
- **Connectors** that expose physical layout of the data: partitioning, sorting, grouping, indices that allow to optimize queries

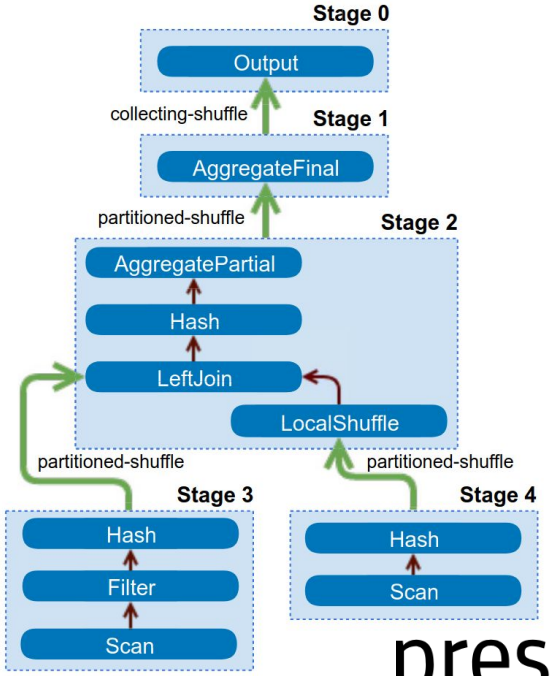


presto

Parallelism

```
SELECT
  orders.orderkey, SUM(tax)
FROM orders
LEFT JOIN lineitem
  ON orders.orderkey = lineitem.orderkey
WHERE discount = 0
GROUP BY orders.orderkey
```

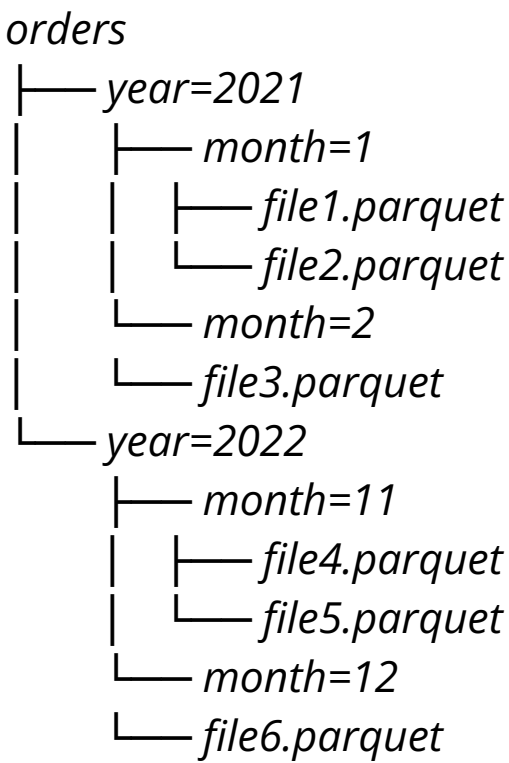
- Presto optimizes query execution with **inter-node** parallelism: executes different parts in a query plan in parallel across workers
- These parts, known as *stages,* are distributed to multiple *tasks*, which execute the same computation on different sets of input data
- In-memory data transfer between stages
- Stages (like connectors) can express properties of their outputs for optimization, partitioning, sorting, etc
- Shuffles are buffered in-memory instead of being persisted to disk, reducing latency
- Also performs **intra-node** parallelism
- *All-at-once* and *phased* scheduling of stages



presto

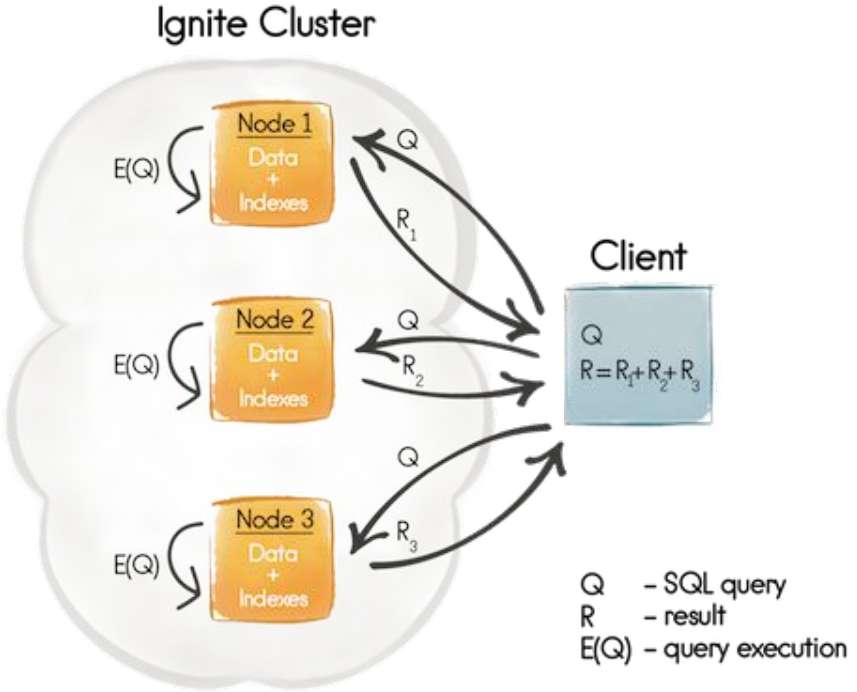
Hive Partitioning

Hive partitioning is a partitioning strategy that is used to split a table into multiple files based on partition keys. The files are organized into folders. Within each folder, the partition key has a value that is determined by the name of the folder.



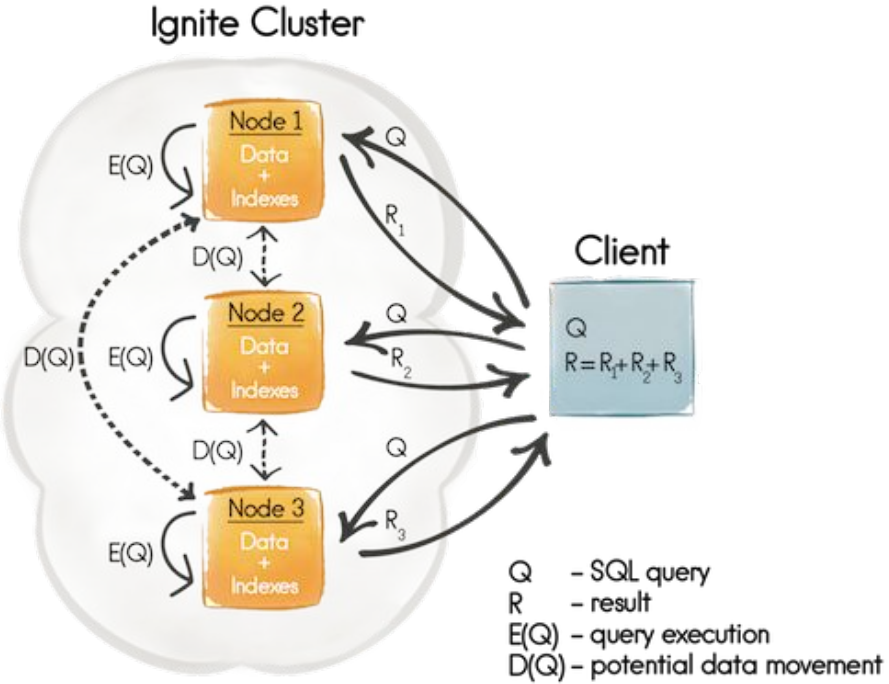
Colocated join

- Tables are **joined on the partitioning column**
- More efficient because they can be efficiently distributed across the cluster nodes
- A colocated join (Q) is sent to all the nodes that store the data matching the query condition
- Then the query is executed over the local data set on each node (E(Q))
- The results (R) are aggregate on the client node that initiated the query



Non-colocated join

- There is **no partitioning on the join column**
- The query is executed locally on nodes with data matching the query condition
- But because the data is not colocated, each node will request missing data from other nodes



Resource Management

- Presto optimizes for overall cluster throughput (aggregate CPU utilized for processing data)
- Node-level scheduler additionally optimizes for low turnaround time for computationally inexpensive queries and the fair sharing CPU resources amongst queries with similar CPU requirements
- The CPU scheduler switches to processing other task after 1 second quanta or when output buffers are full, input buffers are empty, or the system is out of memory
- For deciding which task to run next, a multi-level queue is used. As tasks accumulate CPU time they move to higher level. Each level is assigned a fraction of the available CPU time.
- This complies with users expectation that inexpensive queries should be completed quickly, and are less concerned about the turnaround time of larger, computationally-expensive jobs. Running more queries concurrently, even at the expense of more context-switching, results in lower aggregate queue time, since shorter queries exit the system quickly.
- For memory management there are two mechanism to handle nodes with low memory, (i) *spilling* state to disk, and (ii) promotion to the *reserved memory pool* of the query using most of the memory

Fault Tolerance

- Presto does not have built-in fault tolerance for coordinator or worker node crash failures
- Coordinator failures cause the cluster to become unavailable
- Worker node crash failure causes all queries running on that node to fail
- Clients may retry failed queries, but this is bad for ETL
- In the latest version fault tolerance is provided by executing on Spark