Large Scale Data Processing

MapReduce-Hadoop

Dr. Wenceslao PALMA
wenceslao.palma@ucv.cl

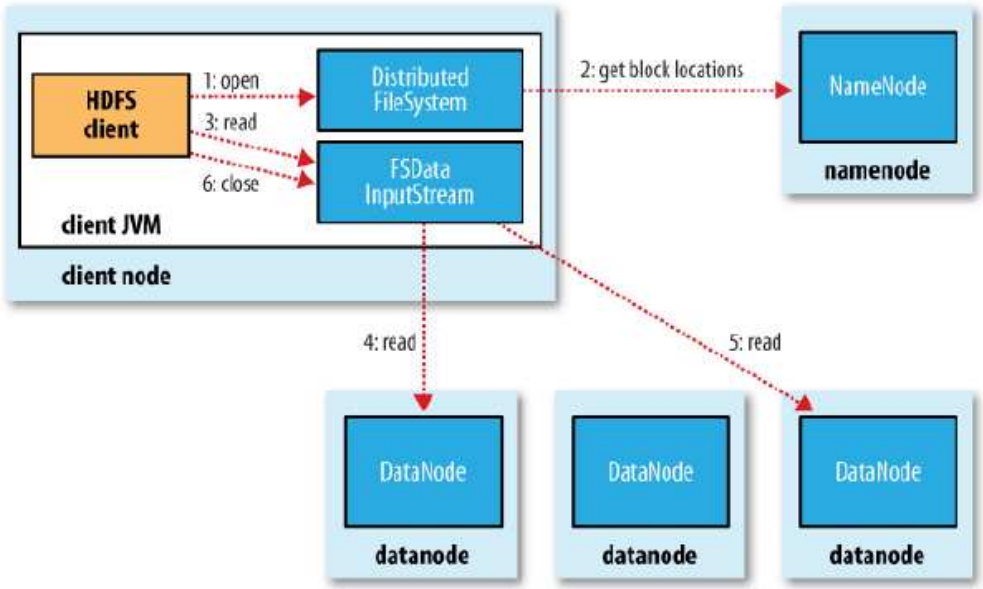
April 2015

Hadoop

Hadoop is an open-source Java-based software plataform developed by the Apache Software Foundation.

- Hadoop implements Google's MapReduce programming model on top of a distributed file system called the Hadoop Distributed File System (HDFS).
- MapReduce divides a job into many tasks.
- HDFS creates multiple replicas of data blocks for reliability.
- Then, MapReduce processes the data where it is located.

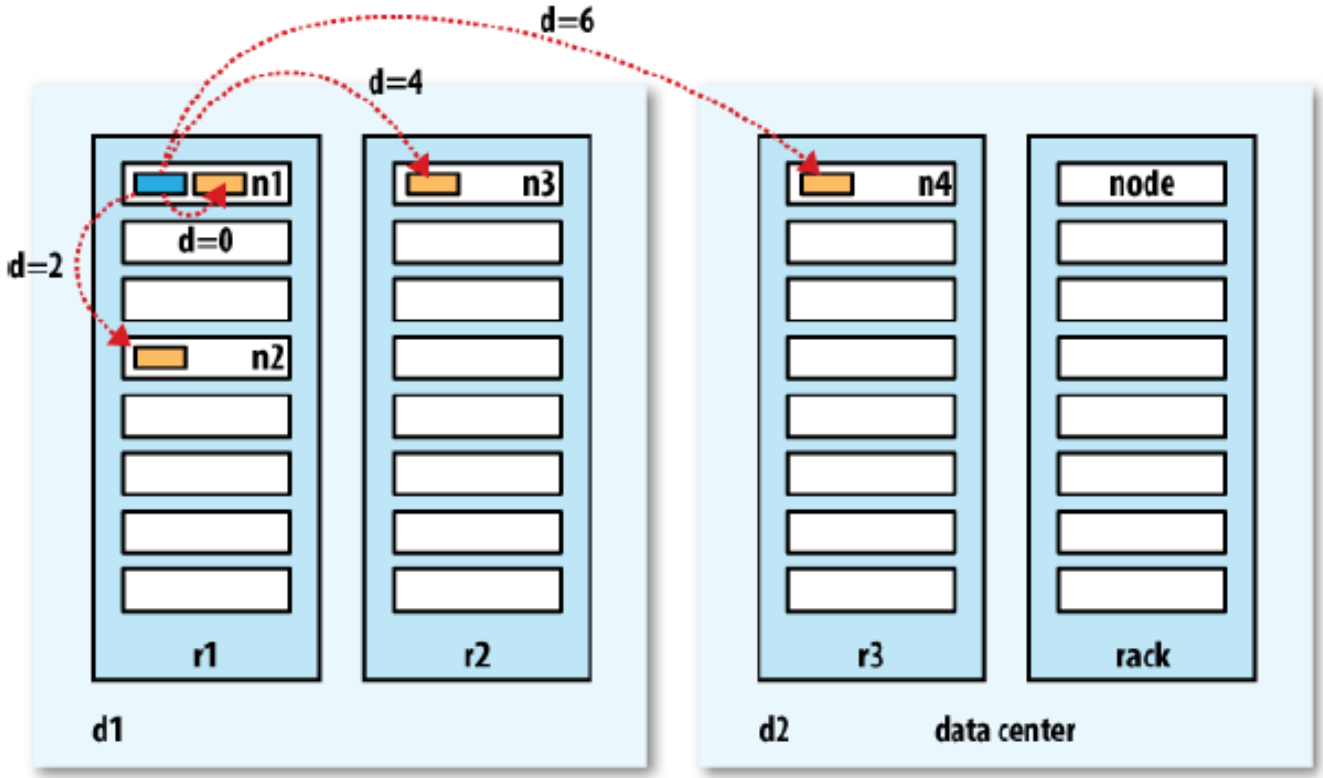
HDFS::Read



- The client node calls the namenode, using RPC, to determine the locations of the blocks.
- For each block, the namenode returns the addresses of the datanodes that have a copy of that block.
- If the client is itself a datanode, then it will read from the local datanode if it stores a copy of the block.

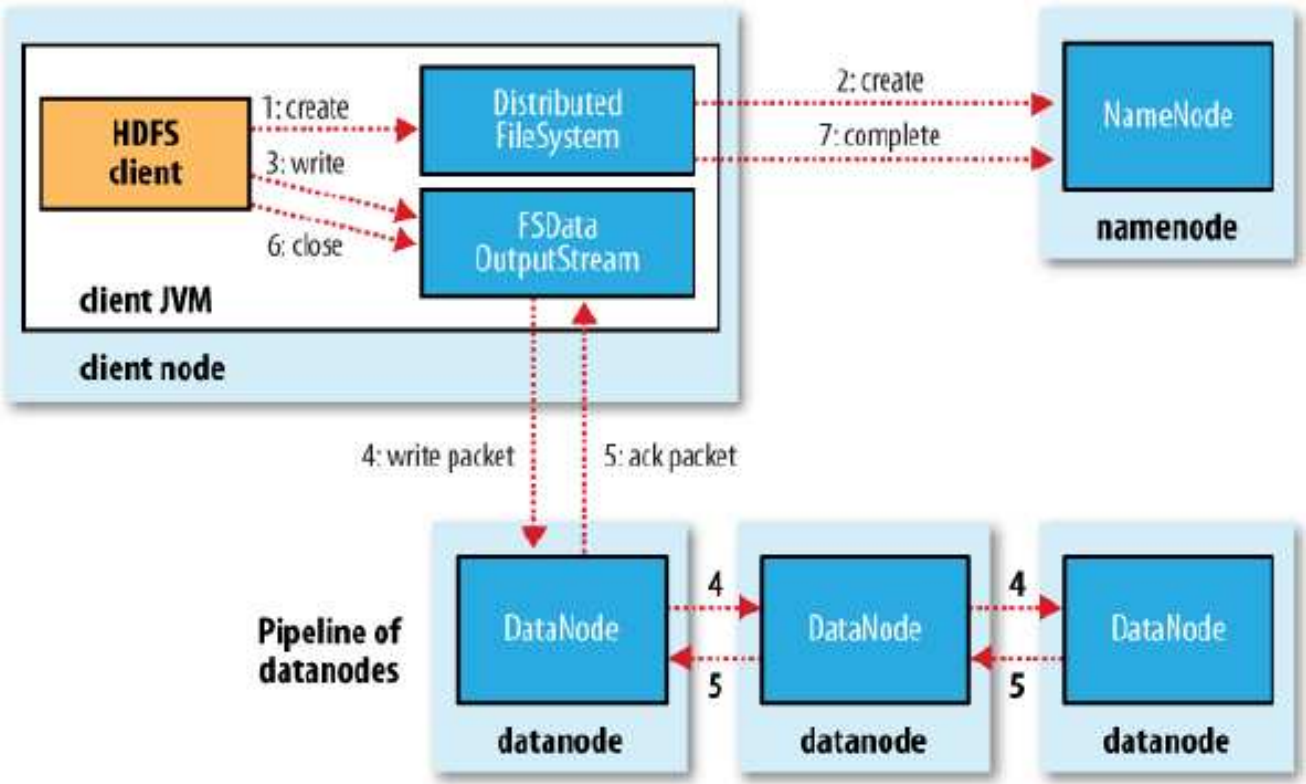
Datanodes are sorted according to their proximity to the client acoording to the topology of the cluster's network

HDFS::Network distance



- The network is represented as a tree
- Distance between two nodes is the sum of their distances the their closest common ancestor.
- $distance(/d1/r1/n1, /d1/r2/n3) = 4$

HDFS::Write



- The client node splits the data into packets which are placed in a data queue.
- Then, it asks the namenode a list of datanodes where to store the replicas.
- The list of nodes forms a pipeline (in the figure the replication level is three).
- An internal queue of packets, called the ack queue, is maintained with packets waiting to be acknowledged.

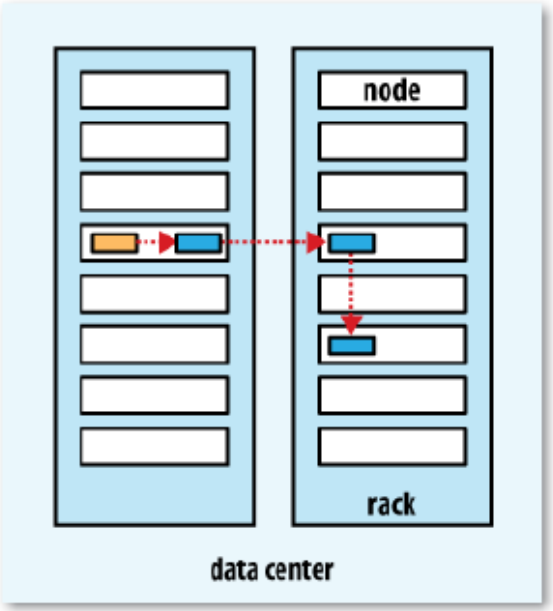
Dealing with errors

- The pipeline is closed and any packets in the ack queue are added to the front of the data queue. Thus, datanodes downstream from the failed node will not miss any packets.
- The current block on the good datanodes is given a new id which is communicated to the namenode. Thus, the partial block on the failed datanode will be deleted when the failed datanode recovers later on.
- When the namenode notices that the block is under-replicated, it creates a further replica on another node.

Fails on multiple datanodes

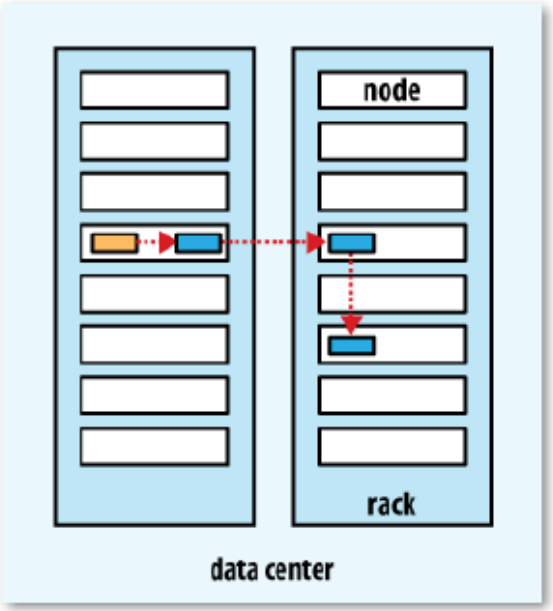
The write succeeds as along as **dfs.replication.min** replicas (default one) are written and the block will be asynchronously replicated until its replication factor **dfs.replication** (default three) is reached.

HDFS::Replica placement



- First replica → on the client node.
- Second replica → is placed on a different rack from the first (chosen at random).
- Third replica → is placed on the same rack but on a different node chosen at random.

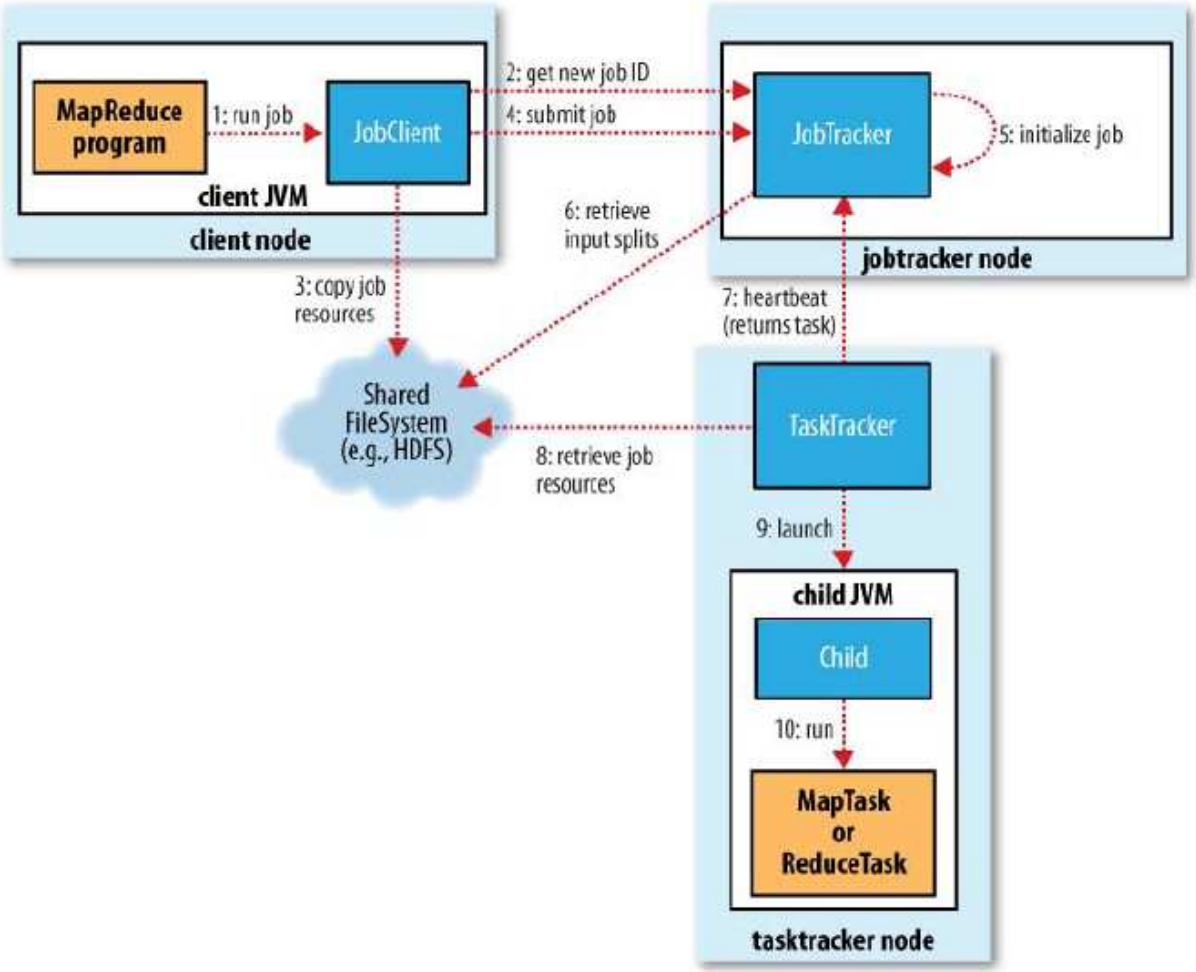
HDFS::Replica placement



- First replica → on the client node.
- Second replica → is placed on a different rack from the first (chosen at random).
- Third replica → is placed on the same rack but on a different node chosen at random.

The replica placement strategy provides a good balance among reliablity, write bandwidth, read performance and block distribution across the cluster

Hadoop::MapReduce job run



The **client** submits the MR job, the **jobtracker** coordinates the job run, the **tasktrackers** run the tasks that the job has been split into, and the **DFS** which is used for sharing job files.

Job submission

- The job submission process asks the jobtracker for a new **JobID**, **checks** the output specification of the job, **computes** the input splits and **copies** the resources needed to run the job.
- The job JAR is copied with a high replication factor (default is 10).
- Finally, it tells the jobtracker that the job is ready for execution.

Job initialization

- During the initialization the jobtracker creates an object to represent the job being run, it encapsulates its tasks and bookkeeping information.
- The lists of tasks to run is created using the input splits. The jobtracker creates one map task for each split.

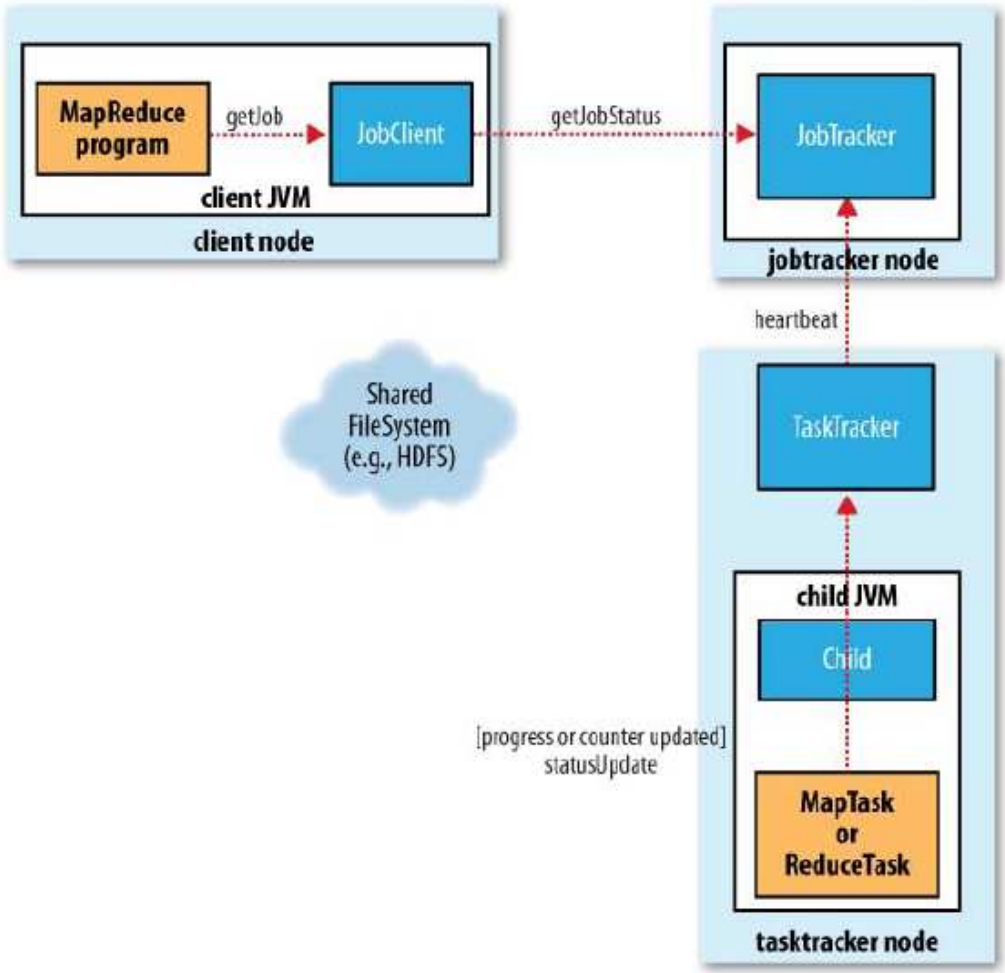
Task assignment

- Tasktrackers send heartbeats to the jobtracker to indicate whether it is ready to run a new task.
- Tasktracker have a fixed number of slots for map tasks and for reduce tasks. The empty map slots are filled before the reduce slots.
- To jobtracker picks the map task whose input split is as close as possible to the tasktracker.

Task execution

- To run a task the tasktracker must localize the job JAR by copying it from the shared filesystem. It creates a local working directory and un-jars the contents of the JAR.
- Then, an instance of TaskRunner is created to run the task.
- The TaskRunner launches a new JVM to run each task.

Hadoop::How status are propagated through MapReduce



Task failure

- When user code in the map or reduce tasks throws a runtime exception the child JVM reports the error to its tasktracker. The tasktracker mark the task as failed freeing up a slot.
- When the tasktracker notices that it has not received a progress update for a while the hanging task is marked as failed. The timeout period is normally 10 minutes. It can be configured in a per job basis by setting **mapred.task.timeout**. It must not be configured to a value of zero.
- When the jobtracker is notified of a failed task, it will reschedule execution of the task.If any task fails four times, the whole job fails. This can be configured using **mapred.map.max.attempts** and **mapred.reduce.max.attempts**.
- However, for some applications it is undesirable to abort the job, as it may be possible to use the results of the job despite some failures. In this case, we can specify the max percentage of tasks that are allowed to fail setting **mapred.max.map.failures.percent** and **mapred.max.reduce.failures.percent**

Tasktracker failure

- When a tasktracker fails or runs very slowly, it will stop sending heartbeats to the jobtracker. The jobtracker notices it and removes it from its pool of tasktrackers. This can be specify setting **mapred.tasktracker.expiry.interval**.
- Any tasks completed successfully or in progress that belongs to an incomplete job are rescheduled.
- A tasktracker can be blacklisted if the number of tasks that have failed on it is significantly higher than an average task failure rate on the cluster.

Jobtracker failure

- It is the most serious failure mode.
- Hadoop, has no mechanism to deal with failure of the jobtracker

Hadoop::Job Scheduling

- Hadoop comes with a choice of schedulers.
- The default is the FIFO queue-based scheduler.
- There are also multiuser schedulers: the Fair Scheduler and the Capacity Scheduler.
- Via the **mapred.job.priority** the priority of a job is set. However, with the FIFO scheduler, priorities do not support preemption.

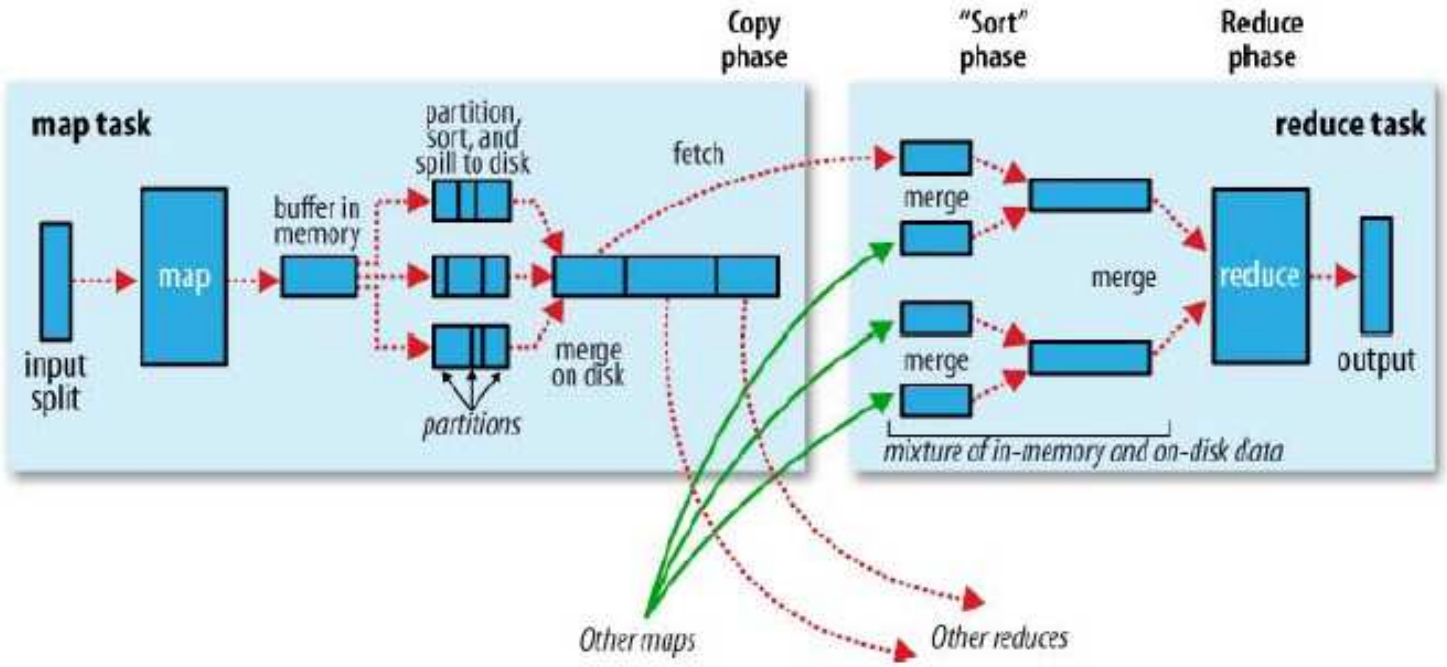
The Fair Scheduler

- A short job belonging to one user will complete in a reasonable time even while another user's long job is running, and the long job will still make progress.
- The Fair Scheduler supports preemption, so if a pool has not received its fair share for a certain period of time, then the scheduler will kill tasks in pools running over capacity in order to give the slots to the pool running under capacity.
- The Fair Scheduler can be enabled setting the property **mapred.jobtracker.taskScheduler** to **org.apache.hadoop.mapred.FairScheduler**

The Capacity Scheduler

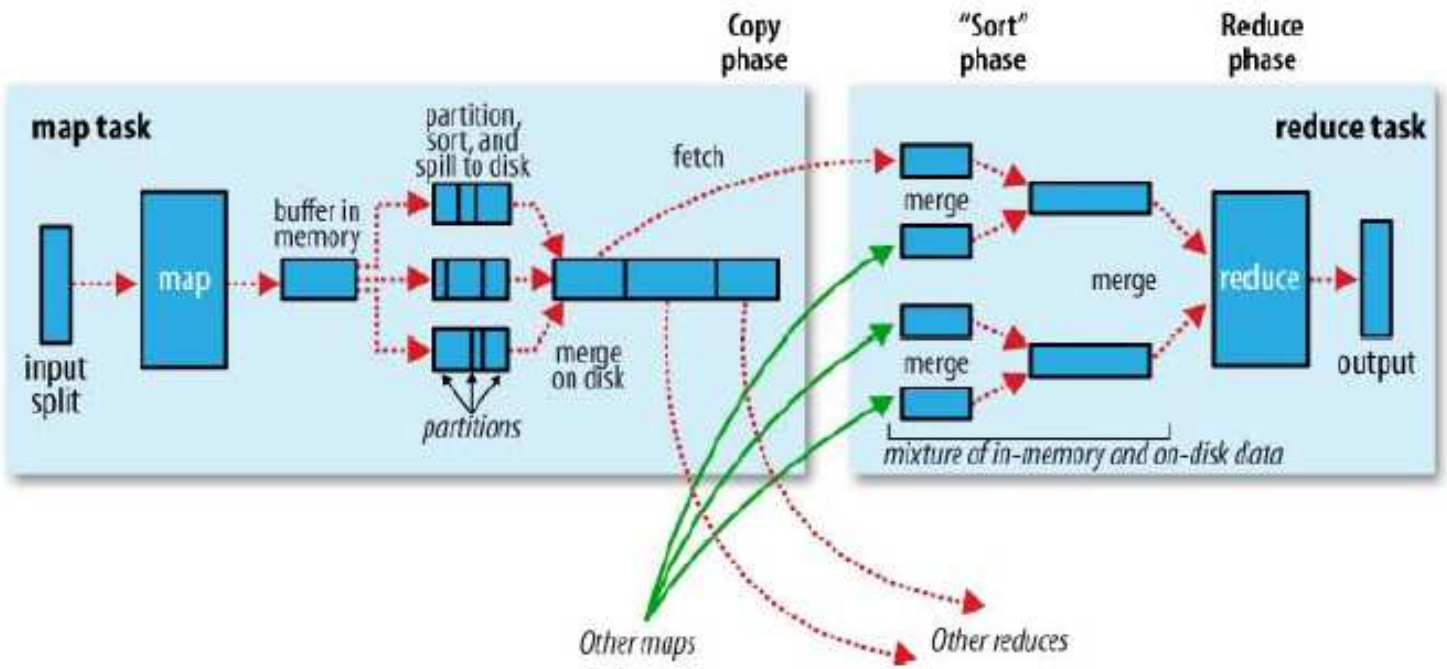
- In the Capacity Scheduler takes a slightly different approach to multiuser scheduling. A cluster is made up of a number of queues, which may be hierarchical (so a queue may be the child of another queue), and each queue has an allocated capacity.
- Within each queue, jobs are scheduled using FIFO scheduling (with priorities).
- The Capacity Scheduler allows users or organizations (defined using queues) to simulate a separate MapReduce cluster with FIFO scheduling for each user or organization.

Hadoop::Shuffle and Sort



- Each map task has a circular memory buffer which is 100 MB by default (see **io.sort.mb** property). When the content of the buffer reaches a certain threshold size (**io.sort.spill.percent**) a background thread spill the contents to disk.
  Before it writes to disk, the thread divides the data into partitions corresponding to the reducers
- By default, the output is not compressed, but it is easy to enable by setting **mapred.compress.map.output** to true.

Hadoop::Shuffle and Sort



- The reduce task starts copying their outputs as soon as each completes.
- The reduce task has a small number of copier threads so that it can fetch map outputs in parallel. The default is five threads, but this number can be changed by setting the **mapred.reduce.parallel.copies** property.
- How do reducers know which tasktrackers to fetch map output from?

Speculative execution

- Hadoop doesn't try to diagnose and fix slow-running tasks; instead, it tries to detect when a task is running slower than expected and launches another, equivalent, task as a backup. This is termed speculative execution of tasks.
- If the original task completes before the speculative task, then the speculative task is killed; on the other hand, if the speculative task finishes first, then the original is killed.
- Speculative execution is turned on by default.

Task JVM Reuse

- Jobs that have a large number of very short-lived tasks (these are usually map tasks), or that have lengthy initialization, can see performance gains when the JVM is reused for subsequent tasks.
- With task JVM reuse enabled, tasks do not run concurrently in a single JVM. The JVM runs tasks sequentially.