Based on the slides of prof. Pietro Michiardi “Hadoop Internals”

Terminology - Recap

- Job: execution of a MapReduce application across a data set
- Task: execution of a Mapper or a Reducer on a split of data
- Task Attempt: attempt to execute a task

For instance, consider running “Word Count” across 20 splits
- 1 job
  - 20 map tasks (one for each input split)
  - A user specified number of reduce tasks
  - At least 20 mapper tasks + number of reducers tasks attempts will be performed
  - More if a machine crashes
Task Attempts
- Each task is attempted at least a maximum number of times (the maximum number of attempts per task is a parameter of the cluster configuration)
- If there is a temporary fault, the execution of each task may initially fail but it succeeds in the following attempts

Multiple attempts may occur in parallel (a.k.a. speculative execution)
- If there is enough available resources (i.e., there are processors in the idle state and enough main memory to run new tasks) Hadoop can duplicate a task and execute each “copy” of the task in a different node of the cluster (containing the input split)
  - Useful if one node has some problems during the execution of the task
  - The maximum number of duplicates per task is equal to the number of replicas of the HDFS file system

Multiple attempts may occur in parallel (a.k.a. speculative execution)
- If there is enough available resources (i.e., there are processors in the idle state and enough main memory to run new tasks) Hadoop can duplicate a task and execute each “copy” of the task in a different node of the cluster (containing the input split)
  - Useful if one node has some problems during the execution of the task
  - The maximum number of duplicates per task is equal to the number of replicas of the HDFS file system

Job Submission
- JobClient class
  - The “submission” of the job in the Driver creates a new instance of a JobClient
  - Then it calls the submitJob() on this class
- Initial verifications before submitting the Job
  - Is there an output directory?
  - Are there any input splits?
  - Can I copy the JAR of the job to HDFS?
    - i.e., Can I copy/move code to data?

Job Initialization
- The JobTracker
  - Creates an object for the job
  - Encapsulating its tasks
  - Manages tasks’ status
- This is where the scheduling happens
  - JobTracker performs scheduling by maintaining a queue
  - Queuing disciplines are pluggable
Job Initialization

- Compute mappers and reducers
  - JobTracker retrieves input splits
    - Computed by JobClient
  - Determines the number of Mappers based on the number of input splits
  - Reads the configuration information to set the number of Reducers

Scheduling

Task Assignment

- Heartbeat-based mechanism
  - TaskTrackers periodically send heartbeats to the JobTracker
    - It means “TaskTracker is alive”
  - Heartbeat contains also information on availability of the TaskTrackers to execute a task

Task Execution

- Now TaskTrackers can
  - Copy the JAR from HDFS
  - Create a local working directory
  - Create an instance of TaskRunner
  - TaskRunner launches a child java virtual machine (JVM)
    - This prevents bugs from stalling the TaskTracker
    - A new child JVM is created for each input split

Scheduling in detail

- FIFO Scheduler (default in vanilla Hadoop)
  - First-come-first-served
    - Long jobs monopolize the cluster
- Fair Scheduler (default in Cloudera)
  - Every user gets a fair share of the cluster capacity over time
    - Jobs are placed into pools, one for each user
      - Users that submit more jobs have no more resources than others
      - Can guarantee minimum capacity per pool
Processes can crash and machines can fail

Task Failure

- Case 1: map or reduce task throws a runtime exception
  - The child JVM reports back to the parent TaskTracker
  - TaskTracker logs the error and marks the TaskAttempt as failed

- Case 2: Hanging tasks
  - TaskTracker notices no progress updates (timeout = 10 minutes)
  - TaskTracker kills the child JVM
  - JobTracker is notified of a failed task
  - Avoid rescheduling the task on the same TaskTracker
  - If a task fails more than maximum times, it is not rescheduled
  - If any task fails maximum times, the job fails

JobTracker Failure

- Currently, Hadoop has no mechanism for this kind of failure
- In future (and commercial) releases
  - Multiple JobTrackers
  - Use ZooKeeper as a coordination mechanism
  - High Availability

In future (and commercial) releases

- Multiple JobTrackers
- Use ZooKeeper as a coordination mechanism
- High Availability
Shuffle and Sort

- The MapReduce framework guarantees the input to every reducer to be sorted by key
  - The process by which the system sorts and transfers map outputs to reducers is known as shuffle
- Shuffle is the most important part of the framework
  - Good understanding allows optimizing both the framework and the execution time of MapReduce jobs
  - Subject to continuous refinements

Shuffle and Sort: the Map Side

- The output of a map task is not simply written to disk
  - In memory buffering
  - Pre-sorting
- Circular memory buffer
  - 100 MB by default
  - Threshold based mechanism to spill buffer content to disk
  - Map output written to the buffer while spilling to disk
  - If buffer fills up while spilling, the map task is blocked

Shuffle and Sort: the Map Side

- Disk spills
  - Written in round-robin to a local dir
  - Output data is partitioned corresponding to the reducers they will be sent to
  - Within each partition, data is sorted (in-memory)
  - Optionally, if there is a combiner, it is executed just after the sort phase

Shuffle and Sort: the Map Side

- More on spills and memory buffer
  - Each time the buffer is full, a new spill is created
  - Once the map task finishes, there are many spills
  - Such spills are merged into a single partitioned and sorted output file

Details on local spill files
The map output file is located on the local disk of TaskTracker.
Another TaskTracker (in charge of a reduce task) requires input from many other TaskTrackers (that finished their map tasks).
How do reducers know which TaskTrackers to fetch map output from?
- When a map task finishes it notifies the parent TaskTracker.
- The TaskTracker notifies (with the heartbeat mechanism) the JobTracker.
- A thread in the reducer polls periodically the JobTracker.
TaskTrackers do not delete local map output as soon as a reduce task has fetched them.

The map output are copied to the TaskTracker running the reducer in memory (if they fit).
- Otherwise they are copied to disk.
Input consolidation:
- A background thread merges all partial inputs into larger, sorted files.
Sorting the input:
- When all map outputs have been copied a merge phase starts.
- All map outputs are sorted maintaining their sort ordering.