Chapter 6

How MapReduce Works

Learning Objectives

- MapReduce Job Anatomy
- Failures
- Job Scheduling
- Shuffle and Sort
- Task Execution

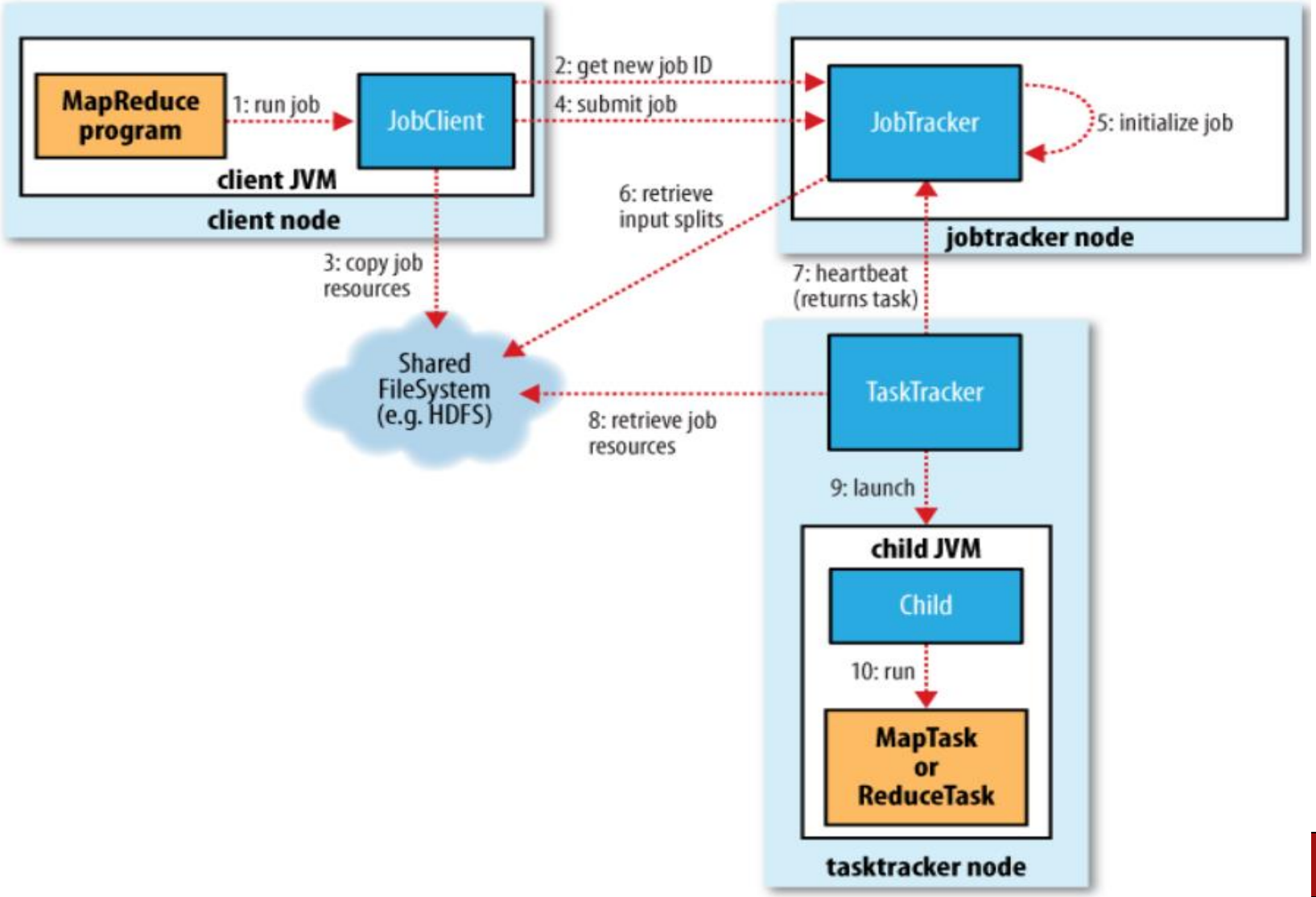
Entities in Classic MapReduce (MapReduce 1)

- **Client** – submits the MapReduce job.
- **Jobtracker** – coordinates the job run; a Java application whose main class is JobTracker.
- **Tasktrackers** – run the tasks that the job has been split into.
- **Distributed file system (HDFS)** – is used for sharing job files between the other entities.

Hadoop – The Definitive Guide

Chapter 6: How MapReduce Works

Classic MapReduce Framework



Job Submission

- Job submission – the submit() method on Job creates an internal JobSummitter instance and calls submitJobInternal() on it, and does the following:
  - Ask the jobtracker for a new job ID;
  - Check the output specification of the job;
  - Computer the input splits for the job;
  - Copy the resources needed to run the job;
  - Tell the jobtracker that the job is ready for execution.

Job Initialization

- Job initialization – the JobTracker puts a job into an internal queue where the job scheduler will pick it up and initialize it. It involves:
  - Creating an object to represent the job being run;
  - Bookkeeping information to keep track of the status and progress of its tasks.

Job Assignment

- Job assignment – the JobTracker assign a task to a TaskTracker when it is ready to run a new task. The JobTracker will
  - Choose a job to select the task from;
  - Set a fixed number of slots for map tasks and for reduce tasks independently.

Task Execution

- Task execution – the TaskTracker runs the task. The TaskTracker will:
  - Localize the job JAR by copying it from the shared filesystem to the tasktracker's filesystem;
  - Create a local working directory for the task and "un-jars" the contents of the JAR into this directory;
  - Create an instance of TaskRunner to run the task.

Progress and Status Updates

- Progress and status updates – MapReduce jobs are long-running batch jobs, taking anything from minutes to hours to run. So, it is important for the user to get feedback.
  - A job and each of its tasks have a *status*;
  - This includes the state of the job or task (e.g., running, successfully completed, failed) and task counters.

Job Completion

- Job completion – when the JobTracker is notified that the last task for a job is complete, it changes the status for the job to "successful." It will:
  - Print a message to tell the user;
  - Return from the waitForCompletion() method;
  - Job statistics and counters are printed to the console.

YARN (MapReduce 2) – 1

- MapReduce 1 hits scalability bottlenecks for a very large cluster of 4,000 nodes or more.
- YARN is the next generation of MapReduce, short for Yet Another Resource Negotiator.
- YARN splits the responsibilities of the JobTracker into separate entities for scalability.
- The JobTracker takes care of both job scheduling and task progress monitoring.

YARN (MapReduce 2) – 2

- YARN uses two independent daemons to separate these two roles:
  - A ***resource manager*** to manage the use of resources across the cluster, and
  - An ***application master*** to manage the lifecycle of applications running on the cluster.
- YARN is more general than MapReduce;
- In fact MapReduce is just one type of YARN application.

Hadoop – The Definitive Guide

Chapter 6: How MapReduce Works

YARN Framework

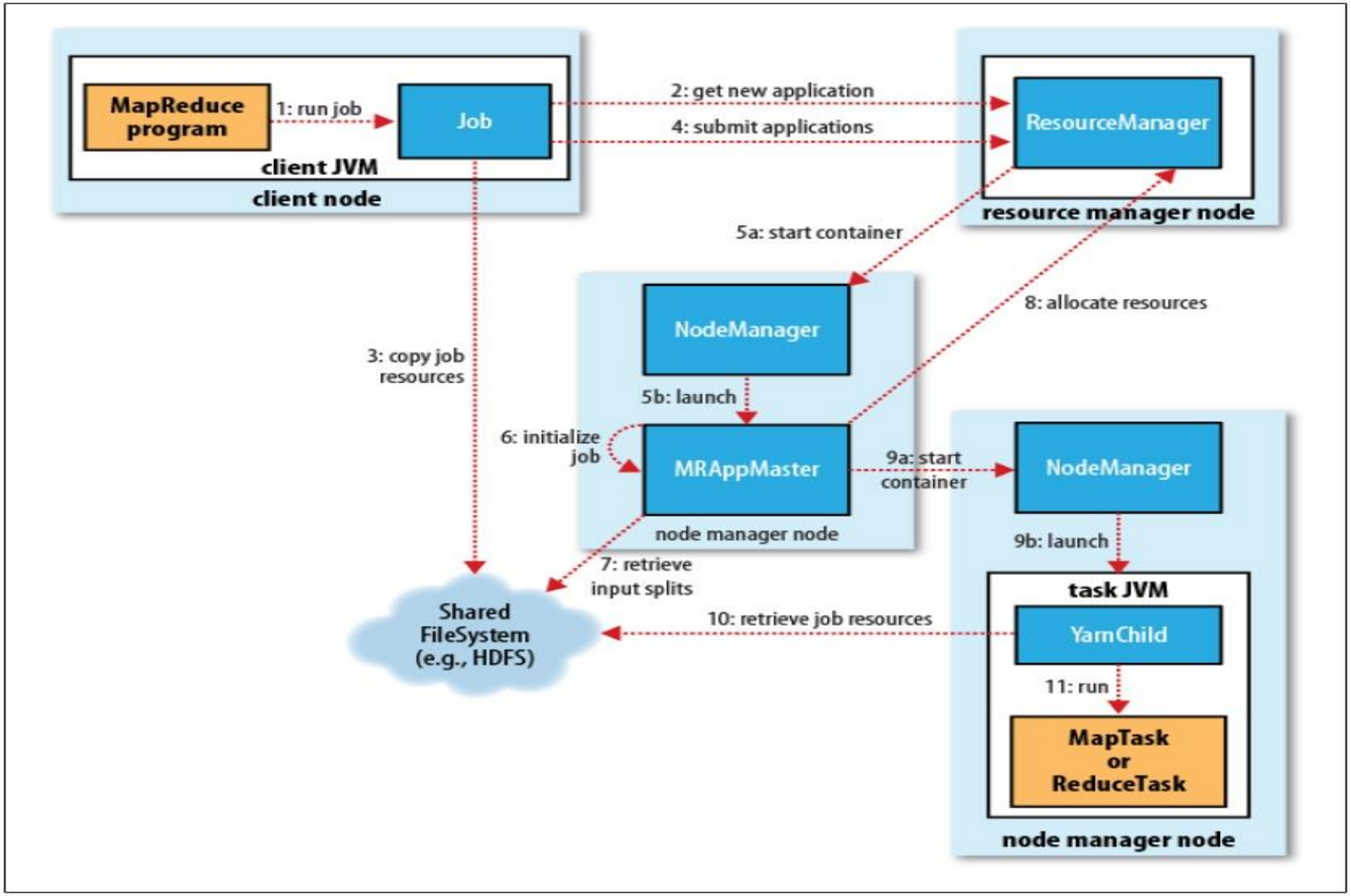


Figure 6-4. How Hadoop runs a MapReduce job using YARN

Failures

There are three failure modes:

- Task failure – when user code in the map or reduce task throws a runtime exception.
- TaskTracker failure – when a TaskTracker fails by crashing or running very slowly and stopping (or very infrequently) sending heartbeats to the JobTracker.
- JobTracker failure – when the JobTracker fails by crashing; all running jobs fail.

Task Failure – 1

- Child task failing
  - Child JVM reports the error back to its parent TaskTracker
- Sudden exit of the child JVM
  - The TaskTracker notices that the process has exited and marks the attempt as failed.
- Hanging tasks
  - The TaskTracker notices that it hasn't received a progress update for a while and mark the task as failed.
  - The child JVM process will be automatically killed after this period (timeout period of 10 minutes)

Task Failure – 2

- Tasktracker
  - Notifying the jobtracker of the failure using heartbeat.
- Jobtracker
  - Reschedule the task on a different TaskTracker.
  - A task failing four or more times will not be retried.

TaskTracker Failure

- Jobtracker
  - Notices a tasktracker that has stopped sending heartbeats;
  - Removes it from tasktracker pool to schedule tasks on;
  - Arranges for map tasks that were run and completed successfully on that tasktracker to be rerun.
- Blacklist
  - A tasktracker is blacklisted if its task failure rate is significantly higher than the average's on the cluster.
  - Blacklisted tasktrackers can be restarted to remove them from the blacklist.

Jobtracker Failure

- Currently, Hadoop has no mechanism for dealing with failure of the jobtracker.
- Jobtracker failure has a low chance of occurring.
- Future work
  - Running multiple jobtrackers, only one of which is the primary jobtracker at any time.
  - Choosing the primary jobtracker using ZooKeeper as a coordination mechanism.

Failures in YARN

Failures in YARN:

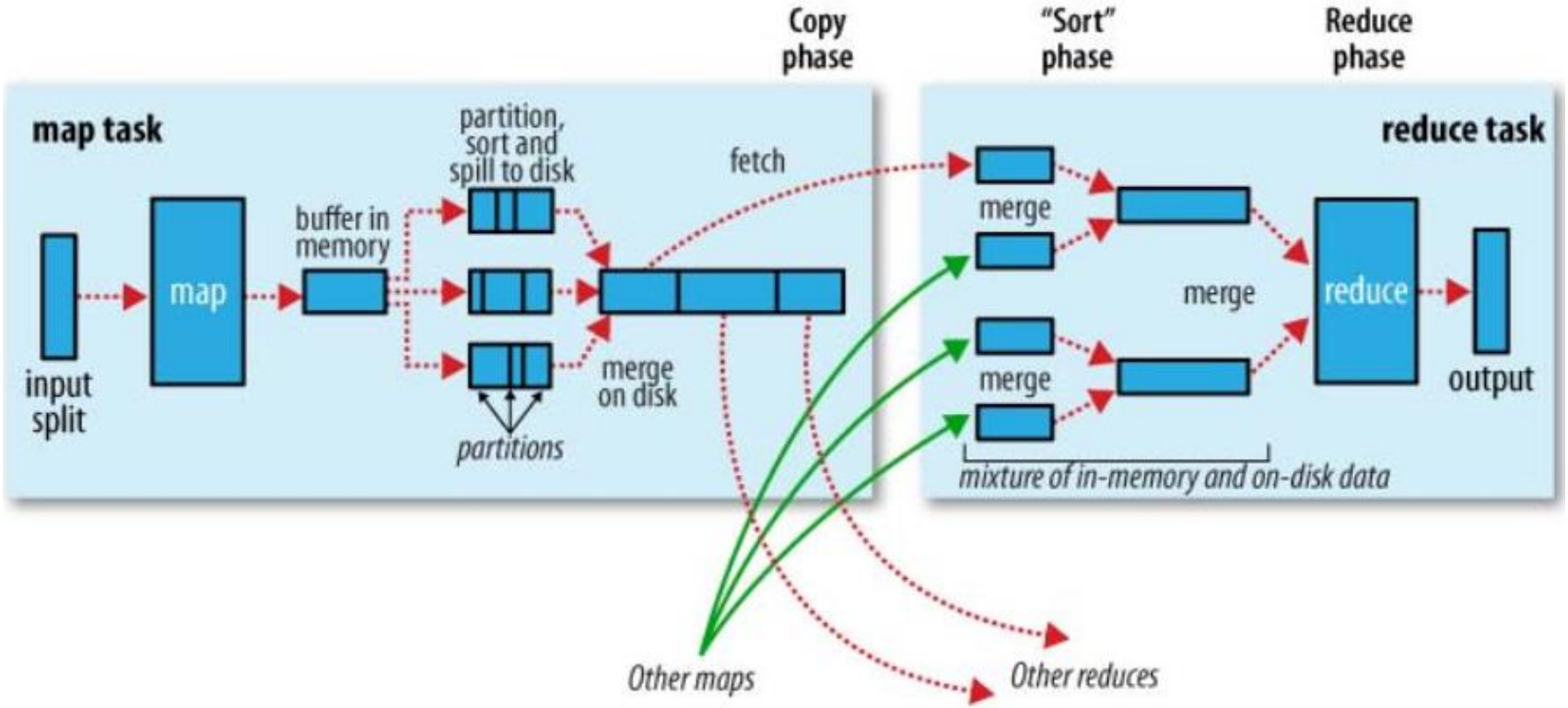
- Task failure – is similar to the classic case.
- Application master failure – give several attempts to succeed; marked as failed if they fail once.
- Node manager failure – resource manager removes node manager from its pool of available nodes.
- Resource manager failure – a new resource manager instance is brought up and it recovers from the saved state.

Job Scheduling

- **FIFO scheduler** (default) – run in order of submission; later on, can set a job's priority, but does not support preemption.
- **Fair scheduler** – each user gets their own pool, where jobs are placed in; a user who submits more jobs will not get any more cluster resources; support preemption.
- **Capacity scheduler** – a cluster is made up of a number of queues, which may be hierarchical; each queue has an allocated capacity; within each queue, jobs are scheduled using FIFO.

Shuffle and Sort

- **Shuffle** is the process that the system performs the **sort** and transfers the map outputs to the reducers as inputs.



The Map Side

- Buffering write and spill to disk
  - Circular memory buffer of 100MB by default;
  - A background thread spills the contents to disk when the contents of the buffer reaches a certain threshold (default: 0.80 = 80%).
  - Before writing to disk, the background thread partitions the data corresponding to the reducers.
  - The thread performs an in-memory sort by key, within each partition.

The Reduce Side

- Copy phase
  - Map tasks may finish at different times.
  - Reduce task starts copying their outputs as soon as each completes.
  - The map outputs also written using memory buffer and spill to disk if it reaches a threshold size.
- Sort phase (merge phase)
  - Map outputs are merged in rounds.

Configuration Tuning

- The general principle is to give the shuffle as much memory as possible.
- For map, the best performance can be obtained by avoiding multiple spills to disk; one is optimal.
- For reduce, the best performance is obtained when the intermediate data can reside entirely in memory.
- Hadoop uses a buffer size of 4 KB by default, which is low, so you should increase this across the cluster.

Speculative Execution

- Job execution time is sensitive to slow-running tasks.
  - Only one straggling task can make the whole job take significantly longer.
- Speculative task – another, equivalent, backup task.
  - Launched only after all the tasks have been launched.
- When a task completes successfully, any duplicate tasks that are running are killed.

Task JVM Reuse

- To reduce the overhead of starting a new JVM for each task. Effective case:
  - Jobs have a large number of very short-lived tasks (these are usually map tasks).
  - Jobs have lengthy initialization.
- If tasktrackers run more than one task at a time, this is always done in separate JVMs.

Skipping Bad Records

- In mapper or reducer code – ignore bad records; throw an exception.
- Using Hadoop's skipping mode – when you can't handle them because there is a bug in a third party library. Skipping process:
  - Task fails
  - Task fails
  - Skipping mode is enabled
    Task fails but failed record is stored by the tasktracker;
  - Skipping mode is still enabled
    Task succeeds by skipping the bad record that failed in the previous attempt.
- Skipping mode can detect only one bad record per task attempt.
- This mechanism is appropriate only for detecting occasional bad records.