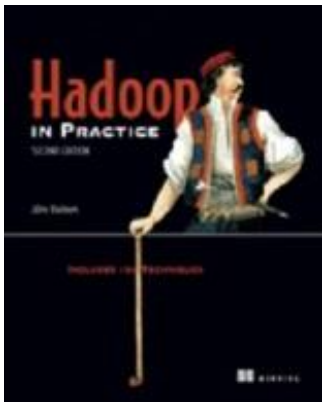
YARN and how MapReduce works in Hadoop

By Alex Holmes



YARN was created so that Hadoop clusters could run any type of work. This meant MapReduce had to become a YARN application and required the Hadoop developers to rewrite key parts of MapReduce. This article will demystify how MapReduce works in Hadoop 2.

Given that MapReduce had to go through some open-heart surgery to get it working as a YARN application, the goal of this article is to demystify how MapReduce works in Hadoop 2.

Dissecting a YARN MapReduce application

Architectural changes had to be made to MapReduce to port it to YARN. Figure 1 shows the processes involved in MRv2 and some of the interactions between them.

Each MapReduce job is executed as a separate YARN application. When you launch a new MapReduce job, the client calculates the input splits and writes them along with other job resources into HDFS (step 1). The client then communicates with the ResourceManager to create the ApplicationMaster for the MapReduce job (step 2). The ApplicationMaster is actually a container, so the ResourceManager will allocate the container when resources become available on the cluster and then communicate with a NodeManager to create the ApplicationMaster container (steps 3–4).[1]

[1] If there aren't any available resources for creating the container, the ResourceManager may choose to kill one or more existing containers to free up space.

The MapReduce ApplicationMaster (MRAM) is responsible for creating map and reduce containers and monitoring their status. The MRAM pulls the input splits from HDFS (step 5) so that when it communicates with the ResourceManager (step 6) it can request that map containers are launched on nodes local to their input data.

Container allocation requests to the ResourceManager are piggybacked on regular heartbeat messages that flow between the ApplicationMaster and the ResourceManager. The heartbeat responses may contain details on containers that are allocated for the application. Data locality is maintained as an important part of the architecture—when it requests map containers, the MapReduce ApplicationManager will use the input splits' location details to request that the containers are assigned to one of the nodes that contains the input splits, and the ResourceManager will make a best attempt at container allocation on these input split nodes.

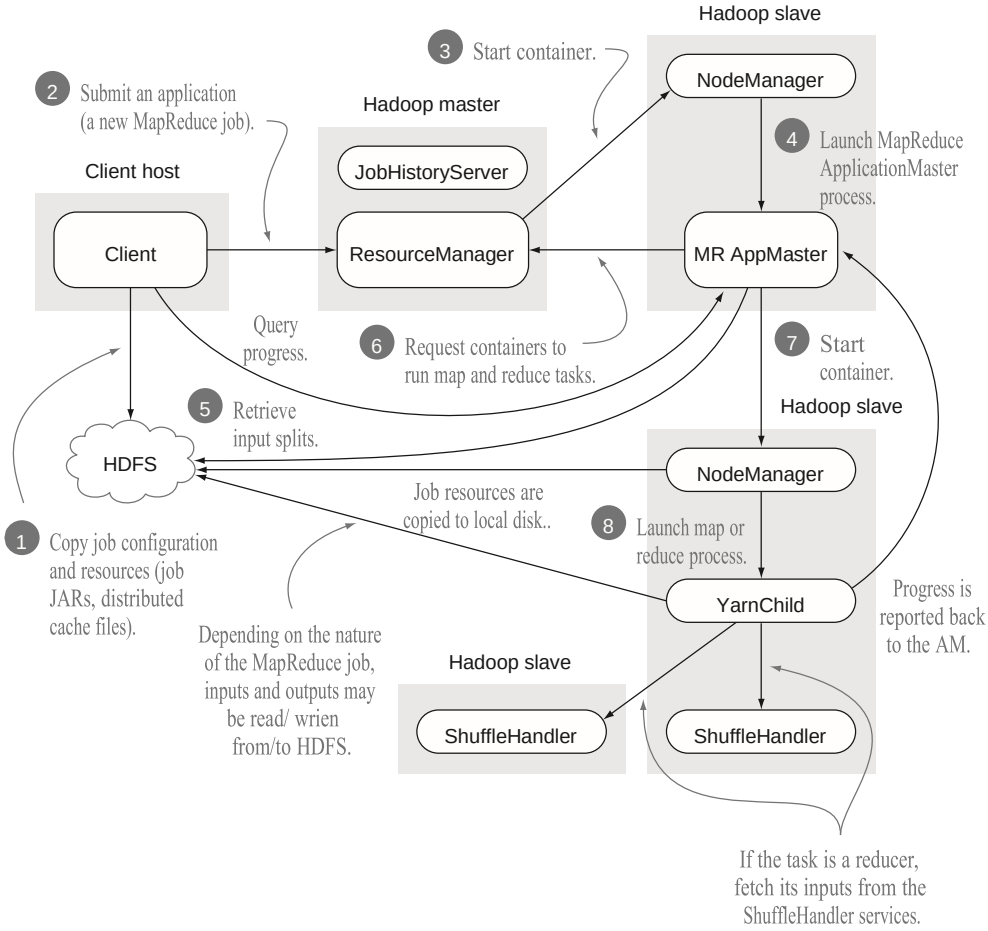


Figure 1 The interactions of a MapReduce 2 YARN application

Once the MapReduce ApplicationManager is allocated a container, it talks to the NodeManager to launch the map or reduce task (steps 7–8). At this point, the map/ reduce process acts very similarly to the way it worked in MRv1.

THE SHUFFLE

The shuffle phase in MapReduce, which is responsible for sorting mapper outputs and distributing them to the reducers, didn't fundamentally change in MapReduce 2. The main difference is that the map outputs are fetched via ShuffleHandlers, which are auxiliary YARN services that run on each slave

node.[2] Some minor memory management tweaks were made to the shuffle implementation; for example, `io.sort.record.percent` is no longer used.

WHERE'S THE JOBTRACKER?

You'll note that the JobTracker no longer exists in this architecture. The scheduling part of the JobTracker was moved as a general-purpose resource scheduler into the YARN ResourceManager. The remaining part of JobTracker, which is primarily the metadata about running and completed jobs, was split in two. Each MapReduce ApplicationMaster hosts a UI that renders details on the current job, and once jobs are completed, their details are pushed to the JobHistoryServer, which aggregates and renders details on all completed jobs.

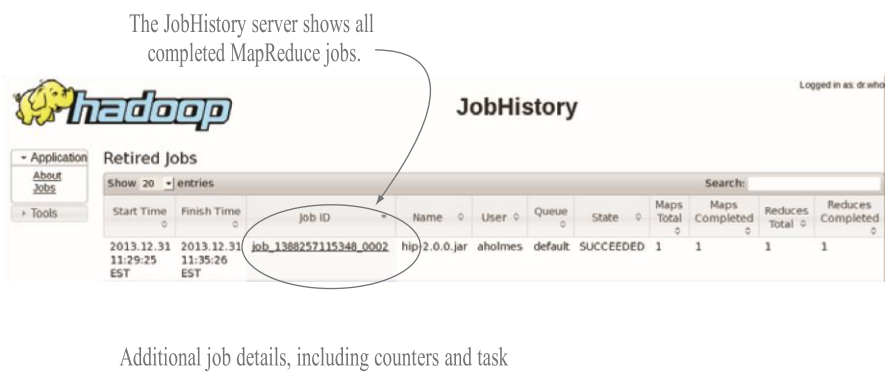
Uber jobs

When running small MapReduce jobs, the time taken for resource scheduling and process forking is often a large percentage of the overall runtime. In MapReduce 1 you didn't have any choice about this overhead, but MapReduce 2 has become smarter and can now cater to your needs to run lightweight jobs as quickly as possible.

TECHNIQUE 7 **Running small MapReduce jobs**

This technique looks at how you can run MapReduce jobs within the MapReduce ApplicationMaster. This is useful when you're working with a small amount of data, as you remove the additional time that MapReduce normally spends spinning up and bringing down map and reduce processes.

TECHNIQUE 7 Running small MapReduce jobs



[2] The ShuffleHandler must be configured in your yarn-site.xml; the property name is `yarn.nodemanager.auxservices` and the `value is mapreduce_shuffle`.

The JobHistory UI, showing MapReduce applications that have completed

■ Problem

You have a MapReduce job that operates on a small dataset, and you want to avoid the overhead of scheduling and creating map and reduce processes.

■ Solution

Configure your job to enable uber jobs; this will run the mappers and reducers in the same process as the ApplicationMaster.

■ Discussion

Uber jobs are jobs that are executed within the MapReduce ApplicationMaster. Rather than liaise with the ResourceManager to create the map and reduce containers, the ApplicationMaster runs the map and reduce tasks within its own process and avoids the overhead of launching and communicating with remote containers.

To enable uber jobs, you need to set the following property: `mapreduce.job.ubertask.enable=true`

Table 1 lists some additional properties that control whether a job qualities for uberization.

Table 1 Properties for customizing uber jobs

Property	Default value	Description
`mapreduce.job.ubertask.maxmaps`	`9`	The number of mappers for a job must be less than or equal to this value for the job to be uberized.
`mapreduce.job.ubertask.maxreduces`	`1`	The number of reducers for a job must be less than or equal to this value for the job to be uberized.
`mapreduce.job.ubertask.maxbytes`	Default block size	The total input size of a job must be less than or equal to this value for the job to be uberized.

When running uber jobs, MapReduce disables speculative execution and also sets the maximum attempts for tasks to `1`.

> **Reducer restrictions** Currently only map-only jobs and jobs with one reducer are supported for uberization.

Uber jobs are a handy new addition to the MapReduce capabilities, and they only work on YARN. This concludes our look at MapReduce on YARN.

To read more about YARN, MapReduce, and Hadoop in action, check out Alex Holmers' book Hadoop in Practice, 2nd edition.