**MapReduce 2.0 (a.k.a. MRv2 or YARN):**

The new architecture divides the two major functions of the JobTracker – resource management and job life-cycle management – into separate components:

* + A **ResourceManager (RM)** that manages the global assignment of compute resources to applications.
  + A per-application **ApplicationMaster (AM)** that manages the application’s life cycle.

In Hadoop 0.23, a MapReduce application is a single job in the sense of classic MapReduce, executed by the MapReduce ApplicationMaster.

There is also a per-machine **NodeManager (NM)** that manages the user processes on that machine. The RM and the NM form the computation fabric of the cluster. The design also allows plugging long-running auxiliary services to the NM; these are application-specific services, specified as part of the configuration, and loaded by the NM during startup. For MapReduce applications on YARN, shuffle is a typical auxiliary service loaded by the NMs. Note that, in Hadoop versions prior to 0.23, shuffle was part of the TaskTracker.

The per-application ApplicationMaster is a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks. In the YARN design, MapReduce is just one application framework; the design permits building and deploying distributed applications using other frameworks. For example, Hadoop 0.23 ships with a Distributed Shell application that permits running a shell script on multiple nodes on the YARN cluster. At the time of writing this blog post, there is also an ongoing development effort to allow running Message Passing Interface (MPI) applications on top of YARN.

**MapReduce 2.0 Design:**

Figure 1 shows a pictorial representation of a YARN cluster. There is a single Resource Manager, which has two main services:

* + A pluggable **Scheduler**, which manages and enforces the resource scheduling policy in the cluster.  Note that, at the time of writing this blog post, there are two schedulers supported in Hadoop 0.23, the default *FIFO* scheduler and the *Capacity* scheduler; the *Fair* Scheduler is not yet supported.
  + An **Applications Manager (AsM)**, which manages running Application Masters in the cluster, i.e., it is responsible for starting application masters and for monitoring and restarting them on different nodes in case of failures.

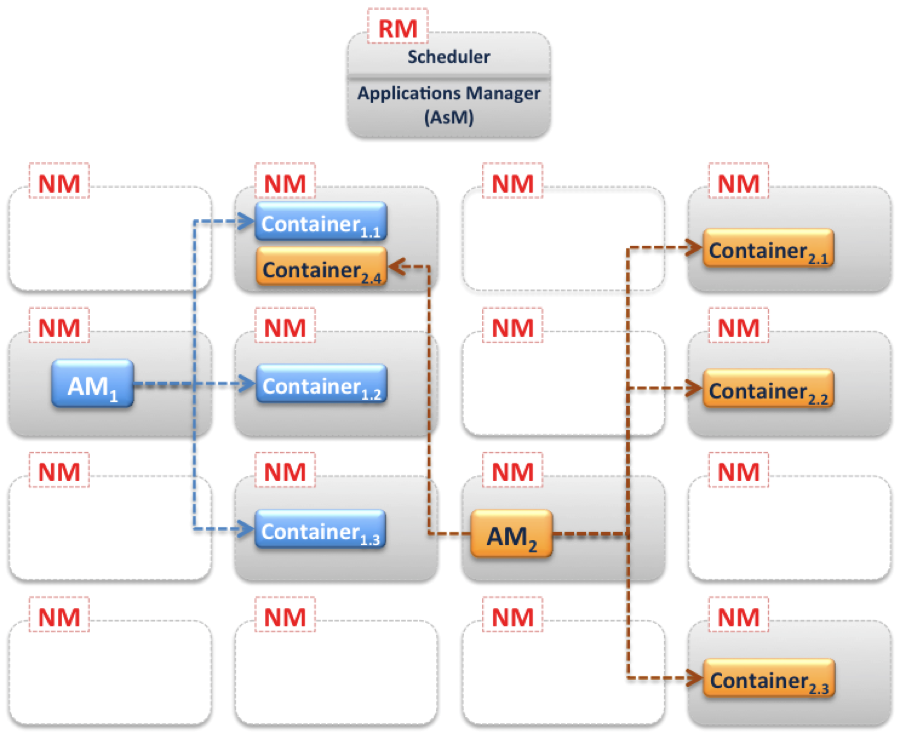


Fig. 1

Figure 1 also shows that there is a NM service running on each node in the cluster. The diagram also shows two AMs (AM1 and AM2). In a YARN cluster at any given time, there will be as many running Application Masters as there are applications (jobs). Each AM manages the application’s(jobs) individual tasks (starting, monitoring and restarting in case of failures). The diagram shows AM1 managing three tasks (containers 1.1, 1.2 and 1.3), while AM2 manages four tasks (containers 2.1, 2.2, 2.3 and 2.4). Each task runs within a Container on each node. The AM acquires such containers from the RM’s Scheduler before contacting the corresponding NMs to start the application’s individual tasks. These **Containers can be roughly compared to Map/Reduce slots** in previous Hadoop versions. However the resource allocation model in Hadoop-0.23 is more optimized from a cluster utilization perspective.

**Resource Allocation Model:**

In earlier Hadoop versions, each node in the cluster was statically assigned the capability of running a predefined number of Map slots and a predefined number of Reduce slots. The slots could not be shared between Maps and Reduces. This static allocation of slots wasn’t optimal since slot requirements vary during the MR job life cycle (typically, there is a demand for Map slots when the job starts, as opposed to the need for Reduce slots towards the end). Practically, in a real cluster, where jobs are randomly submitted and each has its own Map/Reduce slots requirement, having an optimal utilization of the cluster was hard, if not impossible.

The resource allocation model in Hadoop 0.23 addresses such deficiency by providing a more flexible resource modeling. Resources are requested in the form of containers, where each container has a number of non-static attributes. At the time of writing this blog, the only supported attribute was memory (RAM). However, the model is generic and there is intention to add more attributes in future releases (e.g. CPU and network bandwidth). In this new Resource Management model, only a minimum and a maximum for each attribute are defined, and AMs can request containers with attribute values as multiples of these minimums.

**MapReduce 2.0 Main Components:**

In this section, we’ll go through the main components of the new MapReduce architecture in detail to understand the functionality of these components and how they interact with each other.

* + **Client – Resource Manager**

Figure 2 illustrates the initial step for running an application on a YARN cluster. Typically a client communicates with the RM (specifically the Applications Manager component of the RM) to initiates this process. The first step, marked (1) in the diagram, is for the client to notify the Applications Manager of the desire of submitting an application, this is done via a “New Application Request”. The RM respose, marked (2), will typically contain a newly generated unique application ID, in addition to information about cluster resource capabilities that the client will need in requesting resources for running the application’s AM.

Using the information received from the RM, the client can construct and submit an “Application Submission Context”, marked (3), which typically contains information like scheduler queue, priority and user information, in addition to information needed by the RM to be able to launch the AM. This information is contained in a “Container Launch Context”, which contains the application’s jar, job files, security tokens and any resource requirements.

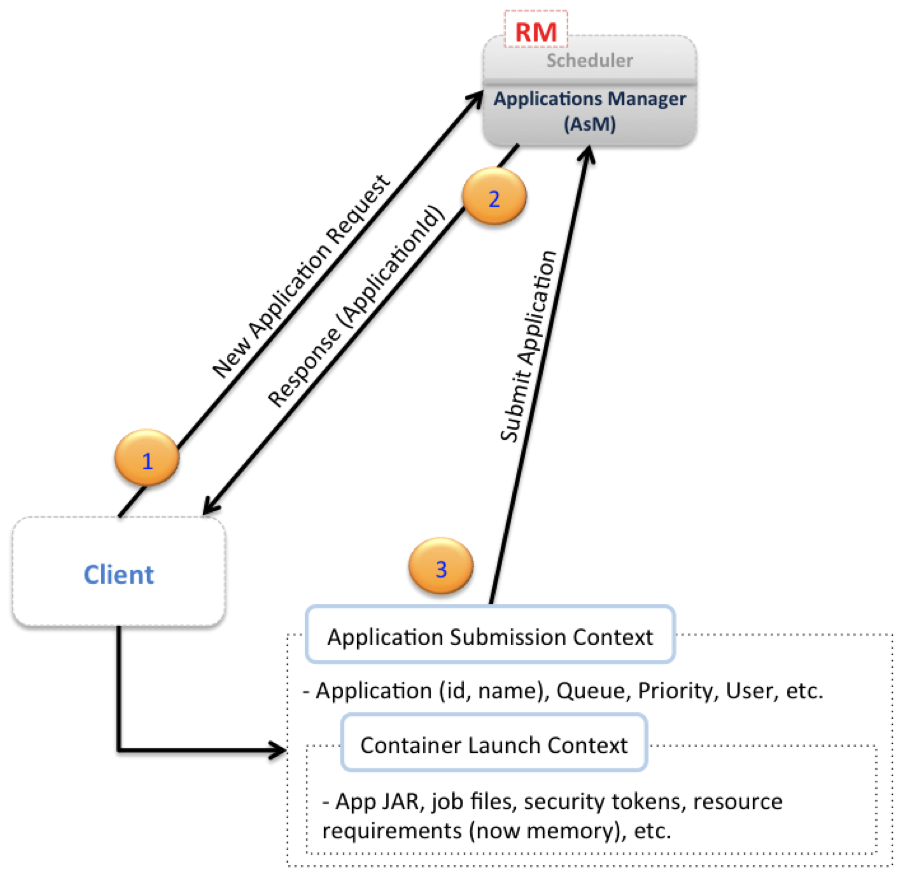


Fig. 2

Following application submission, the client can query the RM for application reports, receive such reports and, if needed, the client can also ask the RM to kill the application. These three additional steps are pictorially depicted in fig. 3.

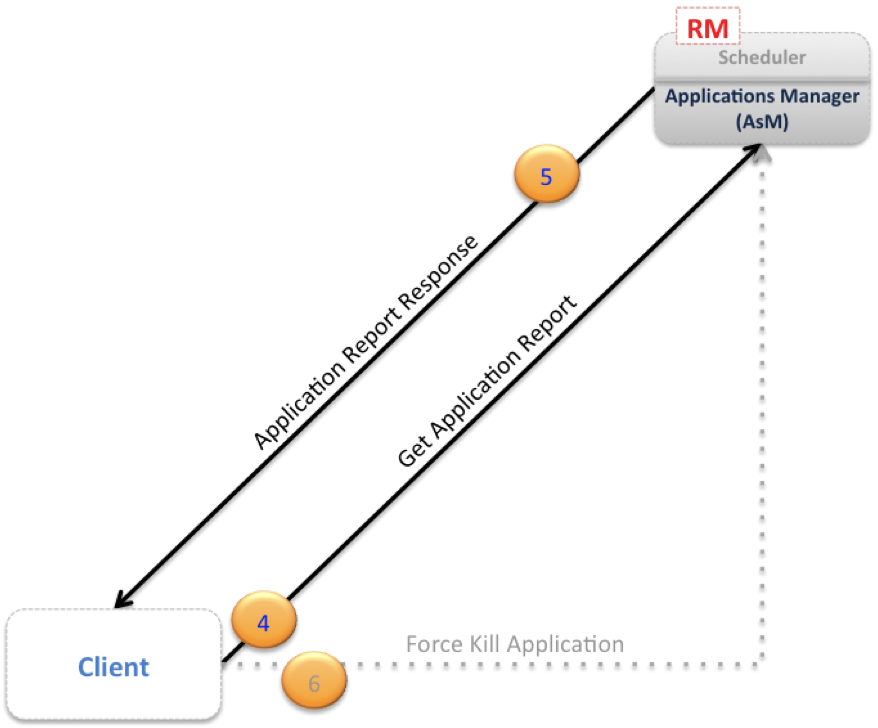


Fig. 3

* + **Resource Manager – Application Master**

When the RM receives the application submission context from the client, it finds an available container meeting the resource requirements for running the AM, and it contacts the NM for the container to start the AM process on this node. Figure 4 depicts the following communication steps between the AM and the RM (specifically the Scheduler component of the RM). The first step, marked (1) in the diagram, is for the AM to register itself with the RM. This step consists of a handshaking procedure and also conveys information like the RPC port that the AM will be listening on, the tracking URL for monitoring the application’s status and progress, etc.

The RM registration response, marked (2), will convey essential information for the AM master like minimum and maximum resource capabilities for this cluster. The AM will use such information in calculating and requesting any resource requests for the application’s individual tasks. The resource allocation request from the AM to the RM, marked (3), mainly contains a list of requested containers, and may also contain a list of released containers by this AM. Heartbeat and progress information are also relayed through resource allocation requests as shown by arrow (4).

When the Scheduler component of the RM receives a resource allocation request, it computes, based on the scheduling policy, a list of containers that satisfy the request and sends back an allocation response, marked (5), which contains a list of allocated resources. Using the resource list, the AM starts contacting the associated node managers (as will be soon seen), and finally, as depicted by arrow (6), when the job finishes, the AM sends a Finish Application message to the Resource Manager and exits.

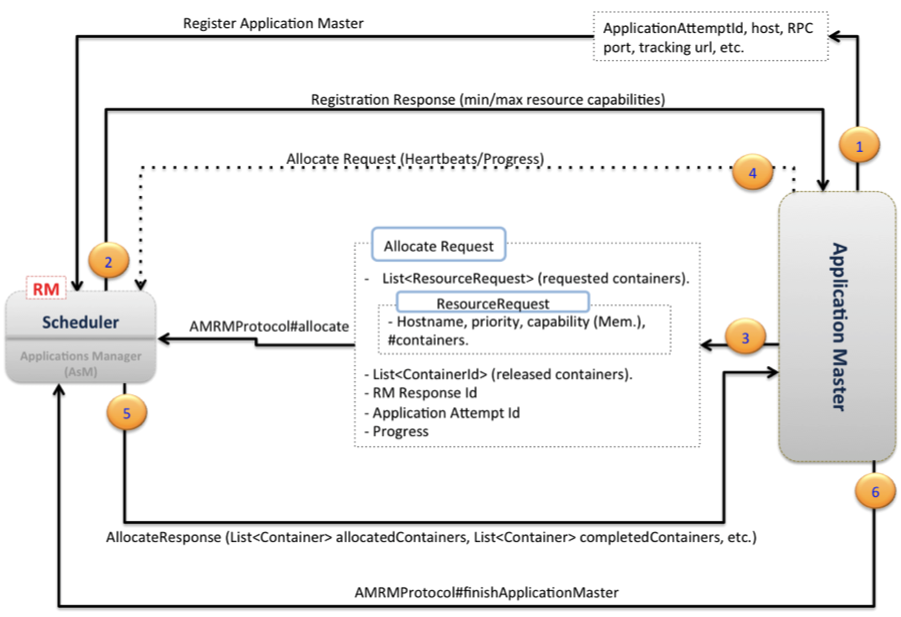


Fig. 4

* + **Application Master – Container Manager**

Figure 5 describes the communication between the AM and the Node Managers. The AM requests the hosting NM for each container to start it as depicted by arrow (1) in the diagram. While containers are running, the AM can request and receive a container status report as shown in steps (2) and (3), respectively.

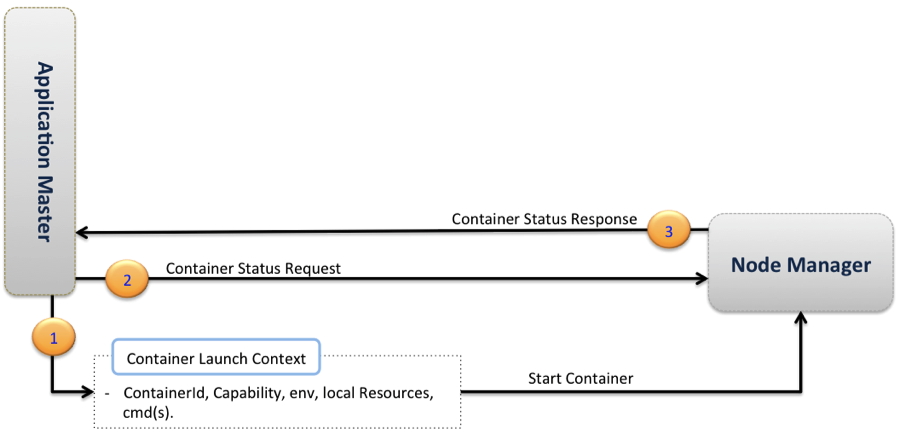


Fig. 5

Based on the above discussion, a developer writing YARN applications will be mainly concerned with the following interfaces:

* + [**ClientRMProtocol**](http://hadoop.apache.org/common/docs/r0.23.0/api/org/apache/hadoop/yarn/api/ClientRMProtocol.html): Client RM (Fig. 3).  
    This is the protocol for a client to communicate with the RM to launch a new application (i.e. an AM), check on the status of the application or kill the application.
  + [**AMRMProtocol**](http://hadoop.apache.org/common/docs/r0.23.0/api/org/apache/hadoop/yarn/api/AMRMProtocol.html): AM RM (Fig. 4).  
    This is the protocol used by the AM to register/unregister itself with the RM, as well as to request resources from the RM Scheduler to run its tasks.
  + [**ContainerManager**](http://hadoop.apache.org/common/docs/r0.23.0/api/org/apache/hadoop/yarn/api/ContainerManager.html): AM NM (Fig. 5).  
    This is the protocol used by the AM to communicate with the NM to start or stop containers and to get status updates on its containers.

**Migrating older MapReduce applications to run on Hadoop 0.23:**

All client-facing MapReduce interfaces are unchanged, which means that there is no need to make any source code changes to run on top of Hadoop 0.23.