PREV
2. Apache Hadoop YARN Install Quick Start

NEXT
4. Functional Overview of YARN Components

3. Apache Hadoop YARN Core Concepts

The new Apache Hadoop YARN resource manager is introduced in this chapter. In addition to allowing non-MapReduce tasks to work within a Hadoop installation, YARN ("Yet Another Resource Negotiator") provides several other advantages over the previous version of Hadoop, including better scalability, cluster utilization, and user agility.

YARN also brings with it several new services that separate it from the standard Hadoop MapReduce model. A new ResourceManager acting purely as a resource scheduler is the sole arbitrator of cluster resources. User applications, including MapReduce jobs, ask for specific resource requests via the new ApplicationMaster component, which in turn negotiates with the ResourceManager to create an application container within the cluster.

By incorporating MapReduce as a YARN framework, YARN also provides full backward compatibility with existing MapReduce tasks and applications.

BEYOND MAPREDUCE

The Apache Hadoop ecosystem continues to grow beyond the simple MapReduce job. Although MapReduce remains at the core of many Hadoop 1.0 tasks, the introduction of YARN has expanded the capability of a Hadoop environment to move beyond the basic MapReduce process.

The basic structure of Hadoop with Apache Hadoop MapReduce version 1 (MRv1) can be seen in Figure 3.1. The two core services, Hadoop File System (HDFS) and MapReduce, form the basis for almost all Hadoop functionality. All other components are built around these services and must use MapReduce to run Hadoop jobs.

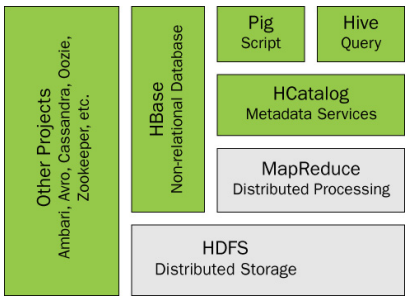


Figure 3.1 The Hadoop 1.0 ecosystem. MapReduce and HDFS are the core components, while other components are built around the core.

Apache Hadoop provides a basis for large-scale MapReduce processing and has spawned a Big Data ecosystem of tools, applications, and vendors. While MapReduce methods enable users to focus on the problem at hand rather than the underlying processing mechanism, they do limit some of the problem domains that can run in the Hadoop framework.

To address these needs, the YARN project was started by the Apache Hadoop community to give Hadoop the ability to run non-MapReduce jobs within the Hadoop framework. YARN provides a generic resource management framework for implementing distributed applications. Starting with Apache Hadoop version 2.0, MapReduce has undergone a complete overhaul; it is now architected as an application on YARN to be called MapReduce version 2 (MRv2). YARN provides both full compatibility with existing MapReduce applications and new support for virtually any distributed application. Figure 3.2 illustrates how YARN fits into the new Hadoop ecosystem.

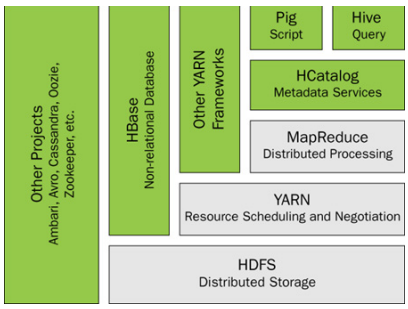


Figure 3.2 YARN adds a more general interface to run non-MapReduce jobs within the Hadoop framework

The introduction of YARN does not change the ability of Hadoop to run MapReduce jobs. It does, however, position MapReduce as merely one of the application frameworks within Hadoop, which works the same way as it did in MRv1. The new capability offered by YARN is the use of new non-MapReduce frameworks that add many new features to the Hadoop ecosystem.

The MapReduce Paradigm

The MapReduce processing model consists of two separate steps. The first step is an embarrassingly parallel map phase, in which input data is split into discrete chunks that can be processed independently. The second step is a reduce phase, in which the output of the map phase is aggregated to produce the desired result. The simple, and fairly restricted, nature of the programming model lends itself to very efficient and extremely large-scale implementations across thousands of low-cost commodity servers (or nodes). When MapReduce is paired with a distributed file system such as Apache Hadoop HDFS, which can provide very high aggregate I/O bandwidth across a large cluster of commodity servers, the economics of the system become extremely compelling—a key factor in the popularity of Hadoop.

One of the keys to Hadoop performance is the *lack of data motion*, such that compute tasks are moved to the servers on which the data reside and not the other way around (i.e., large data movement to compute servers is minimized or eliminated). Specifically, the MapReduce tasks can be scheduled on the same physical nodes on which data reside in HDFS, which exposes the underlying storage layout across the cluster. This design significantly reduces the network I/O patterns and keeps most of the I/O on the local disk or on a neighboring server within the same server rack.

APACHE HADOOP MAPREDUCE

To understand the new YARN process flow, it is helpful to review the original Apache Hadoop MapReduce design. As part of the Apache Software Foundation, Apache Hadoop MapReduce has evolved and improved as an open-source project. This project is an implementation of the MapReduce programming paradigm described previously. The Apache Hadoop MapReduce project itself can be broken down into the following major facets:

- The end-user MapReduce API for programming the desired MapReduce application
- The MapReduce run-time, which is the implementation of various phases such as the map phase, the sort/shuffle/merge aggregation, and the reduce phase
- The MapReduce framework, which is the back-end infrastructure required to run the user's MapReduce application, manage cluster resources, and schedule thousands of concurrent jobs, among other things

This separation of concerns has significant benefits, particularly for end-users where they can completely focus on their application via the API and let the combination of the MapReduce run-time and the framework deal with the complex details such as resource management, fault tolerance, and scheduling.

The current Apache Hadoop MapReduce system is composed of several high-level elements, as shown in Figure 3.3. The master process is the JobTracker, which serves as the clearinghouse for all MapReduce jobs on in the cluster. Each node has a TaskTracker process that manages tasks on the individual node. The TaskTrackers communicate with and are controlled by the JobTracker.

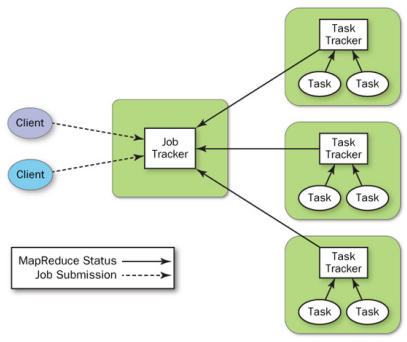


Figure 3.3 Current Hadoop MapReduce control elements

The JobTracker is responsible for *resource management* (managing the worker server nodes—that is, the TaskTrackers), *tracking resource consumption/availability*, and *job life-cycle management* (e.g., scheduling individual tasks of the job, tracking progress, providing fault tolerance for tasks).

The TaskTracker has simple responsibilities—launch/tear down tasks on or-

ders from the JobTracker and provide task status information to the JobTracker periodically.

As described in Chapter 1, the Apache Hadoop MapReduce framework has exhibited some growing pains. In particular, with regard to the JobTracker, several aspects, including scalability, cluster utilization, ability of users to control upgrades to the stack (i.e., user agility), and support for workloads other than MapReduce itself, have been identified as desirable features.

The Need for Non-MapReduce Workloads

MapReduce is great for many applications, but not everything; other programming models better serve requirements such as graph processing (e.g., Google Pregel/Apache Giraph) and iterative modeling using the Message Passing Interface (MPI). As is often the case, much of the enterprise data is already available in Hadoop HDFS, and having multiple paths for processing is critical and a clear necessity. Furthermore, given that MapReduce is essentially batch oriented, support for real-time and near-real-time processing has become an important issue for the user base. A more robust computing environment within Hadoop will enable organizations to see an increased return on their Hadoop investments by lowering operational costs for administrators, reducing the need to move data between Hadoop HDFS and other storage systems, and providing other such efficiencies.

Addressing Scalability

The processing power available in data centers continues to increase rapidly. As an example, consider the additional hardware capability offered by a commodity server over a three-year period:

- 2009: 8 cores, 16 GB of RAM, 4 × 1 TB disk
- 2012: 16+ cores, 72 GB of RAM, 12 × 3 TB of disk

These new servers are often available at the same price point as those of previous generations. In general, servers are twice as capable today as they were two to three years ago—on every dimension. Apache Hadoop MapReduce is known to scale to production deployments of approximately 5000 server nodes of 2009 vintage. Thus, for the same price, the number of CPU cores, amount of RAM, and local storage available to the user will put continued pressure on the scalability of new Apache Hadoop installations.

Improved Utilization

In the current system, the JobTracker views the cluster as composed of nodes (managed by individual TaskTrackers) with distinct map slots and reduce slots, which are not *fungible*. As discussed in Chapter 1, utilization issues occur because maps slots might be "full," while reduce slots are empty (and vice versa). Improving this situation is necessary to ensure the entire system could be used to its maximum capacity for high utilization and applying resources when needed.

User Agility

In real-world deployments, Hadoop is very commonly offered as a shared, multitenant system. As a result, changes to the Hadoop software stack affect a large cross-section of the enterprise, if not the entire enterprise. Against that backdrop, users are very keen to control upgrades to the software stack, as such upgrades have a direct impact on their applications. Thus, allowing multiple, if limited, number of versions of the MapReduce framework is critical for Hadoop.

APACHE HADOOP YARN

The fundamental idea of YARN is to split the two major responsibilities of the JobTracker—that is, resource management and job scheduling/monitoring—into separate daemons: a global ResourceManager and a per-application ApplicationMaster (AM). The ResourceManager and per-node slave, the NodeManager (NM), form the new, and *generic*, operating system for managing applications in a distributed manner.

The ResourceManager is the ultimate authority that arbitrates division of resources among all the applications in the system. The per-application ApplicationMaster is, in effect, a *framework-specific* entity and is tasked with negotiating for resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the component tasks.

The ResourceManager has a pluggable scheduler component, which is responsible for allocating resources to the various running applications subject to the familiar constraints of capacity, queues, and other factors. The Scheduler is a *pure scheduler* in the sense that it performs no monitoring or tracking of status for the application, offering no guarantees on restarting tasks that are not carried out due to either application failure or hardware failure. The scheduler performs its scheduling function based on the *resource requirements* of an application by using the abstract notion of a *resource container*, which incorporates resource dimensions such as memory, CPU, disk, and network.

The NodeManager is the per-machine slave, which is responsible for launching the applications' containers, monitoring their resource usage (CPU, memory, disk, network), and reporting the same to the ResourceManager.

The per-application ApplicationMaster is responsible for negotiating appropriate resource containers from the Scheduler, tracking their status, and monitoring their progress. From the system perspective, the ApplicationMaster runs as a normal *container*. An architectural view of YARN is provided in Figure 3.4.

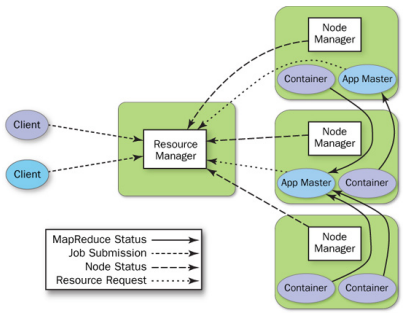


Figure 3.4 New YARN control elements

One of the crucial implementation details for MapReduce within the new YARN system is the reuse of the existing MapReduce framework without any major surgery. This step was very important to ensure compatibility for existing MapReduce applications and users.

YARN COMPONENTS

By adding new functionality, YARN brings new components into the Apache Hadoop workflow. These components provide finer-grained control for the end-user and simultaneously offer more advanced capabilities to the Hadoop ecosystem.

ResourceManager

As mentioned earlier, the YARN ResourceManager is primarily a *pure scheduler*. It is strictly limited to arbitrating requests for available resources in the system made by the competing applications. It optimizes for cluster utilization (i.e., keeps all resources in use all the time) against various constraints such as capacity guarantees, fairness, and service level agreements (SLAs). To allow for different policy constraints, the ResourceManager has a *pluggable scheduler* that enables different algorithms such as those focusing on capacity and fair scheduling to be used as necessary.

ApplicationMaster

An important new *concept* in YARN is the ApplicationMaster. The ApplicationMaster is, in effect, an *instance* of a *framework-specific library* and is responsible for negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the containers and their resource consumption. It has the responsibility of negotiating for appropriate resource containers from the ResourceManager, tracking their status, and monitoring progress.

The ApplicationMaster design enables YARN to offer the following important new features:

■ Scale: The ApplicationMaster provides much of the job-oriented functionality of the JobTracker so that the entire system can scale more dramatically. Simulations have shown that jobs may scale to 10,000 node clusters composed of modern hardware without significant issue. As a pure scheduler, the ResourceManager does not, for example, have to provide fault tolerance for resources across the cluster. By shifting fault tolerance to the ApplicationMaster instance, control becomes local, rather than global. Furthermore, because an instance of an ApplicationMaster is made available per application, the ApplicationMaster itself is rarely a bottleneck in the cluster.

■ Openness: Moving all application framework-specific code into the ApplicationMaster generalizes the system so that it can now support multiple frameworks such as MapReduce, MPI, and Graph Processing.

These features were the result of some key YARN design decisions:

■ Move all complexity (to the extent possible) to the ApplicationMaster, while providing sufficient functionality to allow application framework authors sufficient flexibility and power.

■ Because it is essentially user code, do not trust the ApplicationMaster(s). In other words, no ApplicationMaster is a privileged service.

■ The YARN system (ResourceManager and NodeManager) has to protect itself from faulty or malicious ApplicationMaster(s) and resources granted to them at all costs.

In reality, every application has its own instance of an ApplicationMaster. However, it's completely feasible to implement an ApplicationMaster to manage a set of applications (e.g., ApplicationMaster for Pig or Hive to manage a set of MapReduce jobs). Furthermore, this concept has been stretched to manage long-running services, which manage their own applications (e.g., launching HBase in YARN via a special HBaseAppMaster).

Resource Model

YARN supports a very general resource model for applications. An application (via the ApplicationMaster) can request resources with highly specific requirements, such as the following:

■ Resource name (including hostname, rackname, and possibly complex network topologies)

■ Amount of memory

■ CPUs (number/type of cores)

■ Eventually resources such as disk/network I/O, GPUs, and more

ResourceRequests and Containers

YARN is designed to allow individual applications (via the ApplicationMaster) to utilize cluster resources in a shared, secure, and multitenant manner. It also remains aware of cluster topology so that it can efficiently schedule and optimize data access (i.e., reduce data motion for applications to the extent possible).

To meet those goals, the central Scheduler (in the ResourceManager) maintains extensive information about an application's resource needs, which allows it to make better scheduling decisions across all applications in the cluster. This leads us to the ResourceRequest and the resulting container.

Essentially, an application can ask for specific resource requests via the ApplicationMaster to satisfy its resource needs. The Scheduler responds to a resource request by allocating a container, which satisfies the requirements laid out by the ApplicationMaster in the initial ResourceRequest.

A ResourceRequest has the following form:

Click here to view code image

```
<resource-name, priority, resource-requirement, number-of-containers>
```

These components are described as follows:

- `resource-name` is either hostname, rackname where the resource is desired, or * to indicate no preference. Future plans may support even more complex topologies for virtual machines on a host, more complex networks, and other features.

- `priority` is intra-application priority for this request (not across multiple applications). This orders various ResourceRequests within a given application.

- `resource-requirement` is the required capabilities such as the amount of memory or CPU time (currently YARN supports only memory and CPU).

- `number-of-containers` is just a multiple of such containers. It limits the total number of containers as specified in the ResourceRequest.

Essentially, the container is the resource allocation, which is the successful result of the ResourceManager granting a specific ResourceRequest. A container grants rights to an application to use a specific amount of resources (e.g., memory, CPU) on a specific host.

The ApplicationMaster must take the container and present it to the NodeManager managing the host, on which the container was allocated, to use the resources for launching its tasks. For security reasons, the container allocation is verified, in the secure mode, to ensure that ApplicationMaster(s) cannot fake allocations in the cluster.

Container Specification

While a container, as described previously, is merely the *right* to use a specified amount of resources on a specific machine (NodeManager) in the cluster, the ApplicationMaster must provide considerably more information to the NodeManager to actually *launch* the container. YARN allows applications to launch any process and, unlike existing Hadoop MapReduce, this ability isn't limited to Java applications.

The YARN container launch specification API is platform agnostic and contains the following elements:

- Command line to launch the process within the container

- Environment variables

- Local resources necessary on the machine prior to launch, such as jars, shared-objects, and auxiliary data files

- Security-related tokens

This design allows the ApplicationMaster to work with the NodeManager to launch containers ranging from simple shell scripts to C/Java/Python processes on UNIX/Windows to full-fledged virtual machines.

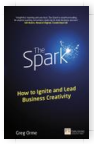
WRAP-UP

The release of Apache Hadoop YARN provides many new capabilities to the existing Hadoop Big Data ecosystem. While the scalable MapReduce paradigm has enabled previously intractable problems to be efficiently managed on large clustered systems, YARN provides a framework for managing both MapReduce and non-MapReduce tasks of greater size and complexity. YARN provides the framework to apply low-cost commodity hardware to virtually any Big Data problem.

PREV
2. Apache Hadoop YARN Install Quick Start

NEXT
4. Functional Overview of YARN Components

BOOK SECTION

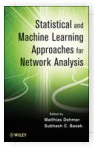


Introduction: The Spark

from: The Spark by Greg Orme
Released: July 2014
10 MINS

Innovation

BOOK SECTION



Tuning Topology Generator Parameters

from: Statistical and Machine Learning Approaches for Network Analysis by Matthias Dehmer...
Released: August 2012
9 MINS

Math & Science

BOOK SECTION

Need for YARN

from: YARN Essentials by Nirmal Kumar...
Released: February 2015
7 MINS

Hadoop

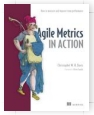
BOOK SECTION

CHAPTER 4: HDFS, Hive, HBase, and HCatalog

from: Microsoft Big Data Solutions by Adam Jorgensen...
Released: March 2014
28 MINS

Hadoop

BOOK SECTION



Chapter 10. Measuring your team against the agile principles

from: Agile Metrics in Action: How to measure and improve team performance by Christopher W. H. Davis
Released: July 2015
40 MINS

Leading Teams

BOOK SECTION

Hadoop and HDInsight in a Heartbeat

from: HDInsight Essentials - Second Edition by Rajesh Nadipalli
Released: January 2015
5 MINS

Hadoop

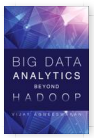
BOOK SECTION

Boosting words closer to each other

from: Solr Cookbook - Third Edition by Rafał Kuć
Released: January 2015
6 MINS

Solr

BOOK SECTION



Introduction: Why Look Beyond Hadoop Map-Reduce?

from: Big Data Analytics Beyond Hadoop: Real-Time Applications with Storm, Spark, and More Hadoop Alternatives by Vijay Srinivas Agneeswaran, Ph.D

Released: May 2014

28 MINS

Hadoop

BOOK SECTION

Chapter 2. Hadoop Fundamental Concepts

from: Virtualizing Hadoop: How to Install, Deploy, and Optimize Hadoop in a Virtualized Architecture by Charles Kim...

Released: July 2015

68 MINS

Hadoop