Learning YARN

The Yet-Another Resource Negotiator (YARN) framework solves the design problems related to resource management faced by the Hadoop 1.x framework by providing a more scalable, efficient, flexible, and highly available resource management framework for distributed data processing.

This book starts with an overview of the YARN features and explains how YARN provides a business solution for growing big data needs. You will learn to provision and manage single, as well as multi-node, Hadoop-YARN clusters in the easiest way. You will walk through the YARN administration, life cycle management, application execution, REST APIs, schedulers, security framework and so on. You will gain insights about the YARN components and features such as ResourceManager, NodeManager, ApplicationMaster, Container, Timeline Server, High Availability, Resource Localisation and so on.

Who this book is written for

This book is intended for those who want to understand what YARN is and how to efficiently use it for the resource management of large clusters. For cluster administrators, this book gives a detailed explanation of provisioning and managing YARN clusters. If you are a Java developer or an open source contributor, this book will help you to drill down the YARN architecture, write your own YARN applications and understand the application execution phases. This book will also help big data engineers explore YARN integration with real-time analytics technologies such as Spark and Storm.

What you will learn from this book

- Explore the YARN features and offerings
- Manage big data clusters efficiently using the YARN framework
- Create single, as well as multi-node, Hadoop-YARN clusters on Linux machines
- Understand the YARN components and their administration
- Gain insights into the application execution flow over a YARN cluster
- Write your own distributed application and execute it over a YARN cluster
- Work with schedulers and queues for the efficient scheduling of applications
- Integrate big data projects such as Spark and Storm with YARN

Akhil Arora         Shrey Mehrotra

In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 1 'Starting with YARN Basics'
- A synopsis of the book’s content
- More information on Learning YARN
About the Authors

**Akhil Arora** works as a senior software engineer with Impetus Infotech and has around 5 years of extensive research and development experience. He joined Impetus Infotech in October 2012 and is working with the innovation labs team. He is a technology expert, good learner, and creative thinker. He is also passionate and enthusiastic about application development in Hadoop and other big data technologies. He loves to explore new technologies and is always ready to work on new challenges. Akhil attained a BE degree in computer science from the Apeejay College of Engineering in Sohna, Haryana, India.

*A beginning for a new voyage, A first step towards my passion and to gain recognition, My first book Learning YARN..!!*

-- Akhil Arora

**Shrey Mehrotra** has more than 5 years of IT experience, and in the past 4 years, he has gained experience in designing and architecting solutions for cloud and big data domains.

Working with big data R&D Labs, he has gained insights into Hadoop, focusing on HDFS, MapReduce, and YARN. His technical strengths also include Hive, PIG, ElasticSearch, Kafka, Sqoop, Flume, and Java. During his free time, he listens to music, watches movies, and enjoys going out with friends.
Preface

Today enterprises generate huge volumes of data. In order to provide effective services and to make smarter and intelligent decisions from these huge volumes of data, enterprises use big data analytics. In recent years, Hadoop is used for massive data storage and efficient distributed processing of data. YARN framework solves design problems faced by Hadoop 1.x framework by providing a more scalable, efficient, flexible, and highly available resource management framework for distributed data processing. It provides efficient scheduling algorithms and utility components for optimized use of resources of cluster with thousands of nodes, running millions of jobs in parallel.

In this book, you’ll explore what YARN provides as a business solution for distributed resource management. You will learn to configure and manage single as well as multi-node Hadoop-YARN clusters. You will also learn about the YARN daemons – ResourceManager, NodeManager, ApplicationMaster, Container, and TimeLine server, and so on.

In subsequent chapters, you will walk through YARN application life cycle management, scheduling and application execution over a Hadoop-YARN cluster. It also covers a detailed explanation of features such as High Availability, Resource Localization, and Log Aggregation. You will learn to write and manage YARN applications with ease.

Toward the end, you will learn about the security architecture and integration of YARN with big data technologies such as Spark and Storm. This book promises conceptual as well as practical knowledge of resource management using YARN.
Preface

What this book covers

Chapter 1, Starting with YARN Basics, gives a theoretical overview of YARN, its background, and need. This chapter starts with the limitations in Hadoop 1.x that leads to the evolution of a resource management framework YARN. It also covers features provided by YARN, its architecture, and advantages of using YARN as a cluster ResourceManager for a variety of batch and real-time frameworks.

Chapter 2, Setting up a Hadoop-YARN Cluster, provides a step-by-step process to set up Hadoop-YARN single-node and multi-node clusters, configuration of different YARN components and an overview of YARN's web user interface.

Chapter 3, Administering a Hadoop-YARN Cluster, provides a detailed explanation of the administrative and user commands provided by YARN. It also provides how to guides for configuring YARN, enable log aggregation, auxiliary services, Ganglia integration, JMX monitoring, and health management, and so on.

Chapter 4, Executing Applications Using YARN, explains the process of executing a YARN application over Hadoop-YARN cluster and monitoring it. This chapter describes the application flow and how the components interact during an application execution in a cluster.

Chapter 5, Understanding YARN Life Cycle Management, gives a detailed description of internal classes involved and their core functionalities. It will help readers to understand internals of state transitions of services involved in the YARN application. It will also help in troubleshooting the failures and examining the current application state.

Chapter 6, Migrating from MRv1 to MRv2, involves the steps and configuration changes required to migrate from MRv1 to MRv2 (YARN). Showcase the enhancements made in MRv2 scheduling, job management, and how to re-use MRv1 jobs in YARN. An introduction to MRv2 components integrated with YARN such as MR Job History Server and Application Master.

Chapter 7, Writing Your Own YARN Applications, describes the steps to write your own YARN applications. This includes Java code snippets for various application components definition and order of execution. It also includes detailed explanation of YARN API for creating YARN applications.

Chapter 8, Dive Deep into YARN Components, provides a detailed description of various YARN components, their roles and responsibilities. It'll also covers an overview of additional features provided by YARN such as resource localization, log management, auxiliary services, and so on.
Chapter 9, *Exploring YARN REST Services*, provides a detailed description of REST-based web services provided by YARN and how we can use the REST services in our applications.

Chapter 10, *Scheduling YARN Applications*, gives a detailed explanation of Scheduler and Queues provided by YARN for better and efficient scheduling of YARN applications. This chapter also covers the limitations of scheduling in Hadoop 1.x and how the new scheduling framework optimizing the cluster resource utilization.

Chapter 11, *Enabling Security in YARN*, explains the component and application-level security provided by YARN. It also gives an overview of YARN security architecture for interprocess, intercomponent communication, and token management.

Chapter 12, *Real-time Data Analytics Using YARN*, explains YARN adoption as a resource manager by various real-time analytics tools such as Apache Spark, Storm, and Giraph.
Starting with YARN Basics

In early 2006, Apache Hadoop was introduced as a framework for the distributed processing of large datasets stored across clusters of computers, using a programming model. Hadoop was developed as a solution to handle big data in a cost effective and easiest way possible. Hadoop consisted of a storage layer, that is, Hadoop Distributed File System (HDFS) and the MapReduce framework for managing resource utilization and job execution on a cluster. With the ability to deliver high performance parallel data analysis and to work with commodity hardware, Hadoop is used for big data analysis and batch processing of historical data through MapReduce programming.

With the exponential increase in the usage of social networking sites such as Facebook, Twitter, and LinkedIn and e-commerce sites such as Amazon, there was the need of a framework to support not only MapReduce batch processing, but real-time and interactive data analysis as well. Enterprises should be able to execute other applications over the cluster to ensure that cluster capabilities are utilized to the fullest. The data storage framework of Hadoop was able to counter the growing data size, but resource management became a bottleneck. The resource management framework for Hadoop needed a new design to solve the growing needs of big data.

YARN, an acronym for Yet Another Resource Negotiator, has been introduced as a second-generation resource management framework for Hadoop. YARN is added as a subproject of Apache Hadoop. With MapReduce focusing only on batch processing, YARN is designed to provide a generic processing platform for data stored across a cluster and a robust cluster resource management framework.

In this chapter, we will cover the following topics:

- Introduction to MapReduce v1
- Shortcomings of MapReduce v1
- An overview of the YARN components
Introduction to MapReduce v1

MapReduce is a software framework used to write applications that simultaneously process vast amounts of data on large clusters of commodity hardware in a reliable, fault-tolerant manner. It is a batch-oriented model where a large amount of data is stored in Hadoop Distributed File System (HDFS), and the computation on data is performed as MapReduce phases. The basic principle for the MapReduce framework is to move computed data rather than move data over the network for computation. The MapReduce tasks are scheduled to run on the same physical nodes on which data resides. This significantly reduces the network traffic and keeps most of the I/O on the local disk or within the same rack.

The high-level architecture of the MapReduce framework has three main modules:

- **MapReduce API**: This is the end-user API used for programming the MapReduce jobs to be executed on the HDFS data.
- **MapReduce framework**: This is the runtime implementation of various phases in a MapReduce job such as the map, sort/shuffle/merge aggregation, and reduce phases.
- **MapReduce system**: This is the backend infrastructure required to run the user's MapReduce application, manage cluster resources, schedule thousands of concurrent jobs, and so on.

The MapReduce system consists of two components—JobTracker and TaskTracker.

- **JobTracker** is the master daemon within Hadoop that is responsible for resource management, job scheduling, and management. The responsibilities are as follows:
  - Hadoop clients communicate with the JobTracker to submit or kill jobs and poll for jobs' progress
  - JobTracker validates the client request and if validated, then it allocates the TaskTracker nodes for map-reduce tasks execution
  - JobTracker monitors TaskTracker nodes and their resource utilization, that is, how many tasks are currently running, the count of map-reduce task slots available, decides whether the TaskTracker node needs to be marked as blacklisted node, and so on
JobTracker monitors the progress of jobs and if a job/task fails, it automatically reinitializes the job/task on a different TaskTracker node.

JobTracker also keeps the history of the jobs executed on the cluster.

- **TaskTracker** is a per node daemon responsible for the execution of map-reduce tasks. A TaskTracker node is configured to accept a number of map-reduce tasks from the JobTracker, that is, the total map-reduce tasks a TaskTracker can execute simultaneously. The responsibilities are as follows:

  - TaskTracker initializes a new JVM process to perform the MapReduce logic. Running a task on a separate JVM ensures that the task failure does not harm the health of the TaskTracker daemon.
  - TaskTracker monitors these JVM processes and updates the task progress to the JobTracker on regular intervals.
  - TaskTracker also sends a heartbeat signal and its current resource utilization metric (available task slots) to the JobTracker every few minutes.

### Shortcomings of MapReducev1

Though the Hadoop MapReduce framework was widely used, the following are the limitations that were found with the framework:

- **Batch processing only**: The resources across the cluster are tightly coupled with map-reduce programming. It does not support integration of other data processing frameworks and forces everything to look like a MapReduce job. The emerging customer requirements demand support for real-time and near real-time processing on the data stored on the distributed file systems.

- **Non scalability and inefficiency**: The MapReduce framework completely depends on the master daemon, that is, the JobTracker. It manages the cluster resources, execution of jobs, and fault tolerance as well. It is observed that the Hadoop cluster performance degrades drastically when the cluster size increases above 4,000 nodes or the count of concurrent tasks crosses 40,000. The centralized handling of jobs control flow resulted in endless scalability concerns for the scheduler.
• **Unavailability and unreliability:** The availability and reliability are considered to be critical aspects of a framework such as Hadoop. A single point of failure for the MapReduce framework is the failure of the JobTracker daemon. The JobTracker manages the jobs and resources across the cluster. If it goes down, information related to the running or queued jobs and the job history is lost. The queued and running jobs are killed if the JobTracker fails. The MapReduce v1 framework doesn't have any provision to recover the lost data or jobs.

• **Partitioning of resources:** A MapReduce framework divides a job into multiple map and reduce tasks. The nodes with running the TaskTracker daemon are considered as resources. The capability of a resource to execute MapReduce jobs is expressed as the number of map-reduce tasks a resource can execute simultaneously. The framework forced the cluster resources to be partitioned into map and reduce task slots. Such partitioning of the resources resulted in less utilization of the cluster resources.

**hadoop1namenode** **Hadoop Machine List**

<table>
<thead>
<tr>
<th>Task Trackers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>tracker_hadoop1namenode:127.0.0.1:8888</td>
</tr>
</tbody>
</table>

If you have a running Hadoop 1.x cluster, you can refer to the JobTracker web interface to view the map and reduce task slots of the active TaskTracker nodes.

The link for the active TaskTracker list is as follows:
http://JobTrackerHost:50030/machines.jsp?type=active

• **Management of user logs and job resources:** The user logs refer to the logs generated by a MapReduce job. Logs for MapReduce jobs. These logs can be used to validate the correctness of a job or to perform log analysis to tune up the job's performance. In MapReduce v1, the user logs are generated and stored on the local file system of the slave nodes. Accessing logs on the slaves is a pain as users might not have the permissions issued. Since logs were stored on the local file system of a slave, in case the disk goes down, the logs will be lost.
A MapReduce job might require some extra resources for job execution. In the MapReduce v1 framework, the client copies job resources to the HDFS with the replication of 10. Accessing resources remotely or through HDFS is not efficient. Thus, there's a need for localization of resources and a robust framework to manage job resources.

In January 2008, Arun C. Murthy logged a bug in JIRA against the MapReduce architecture, which resulted in a generic resource scheduler and a per job user-defined component that manages the application execution.
You can see this at https://issues.apache.org/jira/browse/MAPREDUCE-279

An overview of YARN components

YARN divides the responsibilities of JobTracker into separate components, each having a specified task to perform. In Hadoop-1, the JobTracker takes care of resource management, job scheduling, and job monitoring. YARN divides these responsibilities of JobTracker into ResourceManager and ApplicationMaster. Instead of TaskTracker, it uses NodeManager as the worker daemon for execution of map-reduce tasks. The ResourceManager and the NodeManager form the computation framework for YARN, and ApplicationMaster is an application-specific framework for application management.

- **ResourceManager**
  - A per-cluster service
  - Authorizes resources among all the applications in the system
  - Has a pluggable scheduler for cluster resource optimization & an ApplicationsManager

- **NodeManager**
  - A per-machine framework agent, the "worker” daemon in YARN
  - Creates applications’ execution container, monitors their resource usage and reports to the ResourceManager

- **ApplicationMaster**
  - A per-application framework specific library
  - Negotiates resources from the ResourceManager
  - Works with the NodeManager(s) to execute and monitor the tasks
ResourceManager

A ResourceManager is a per cluster service that manages the scheduling of compute resources to applications. It optimizes cluster utilization in terms of memory, CPU cores, fairness, and SLAs. To allow different policy constraints, it has algorithms in terms of pluggable schedulers such as capacity and fair that allows resource allocation in a particular way.

ResourceManager has two main components:

- **Scheduler**: This is a pure pluggable component that is only responsible for allocating resources to applications submitted to the cluster, applying constraint of capacities and queues. Scheduler does not provide any guarantee for job completion or monitoring, it only allocates the cluster resources governed by the nature of job and resource requirement.

- **ApplicationsManager (AsM)**: This is a service used to manage application masters across the cluster that is responsible for accepting the application submission, providing the resources for application master to start, monitoring the application progress, and restart, in case of application failure.

NodeManager

The NodeManager is a per node worker service that is responsible for the execution of containers based on the node capacity. Node capacity is calculated based on the installed memory and the number of CPU cores. The NodeManager service sends a heartbeat signal to the ResourceManager to update its health status. The NodeManager service is similar to the TaskTracker service in MapReduce v1. NodeManager also sends the status to ResourceManager, which could be the status of the node on which it is running or the status of tasks executing on it.

ApplicationMaster

An ApplicationMaster is a per application framework-specific library that manages each instance of an application that runs within YARN. YARN treats ApplicationMaster as a third-party library responsible for negotiating the resources from the ResourceManager scheduler and works with NodeManager to execute the tasks. The ResourceManager allocates containers to the ApplicationMaster and these containers are then used to run the application-specific processes. ApplicationMaster also tracks the status of the application and monitors the progress of the containers. When the execution of a container gets complete, the ApplicationMaster unregisters the containers with the ResourceManager and unregisters itself after the execution of the application is complete.
Container
A container is a logical bundle of resources in terms of memory, CPU, disk, and so on that is bound to a particular node. In the first version of YARN, a container is equivalent to a block of memory. The ResourceManager scheduler service dynamically allocates resources as containers. A container grants rights to an ApplicationMaster to use a specific amount of resources of a specific host. An ApplicationMaster is considered as the first container of an application and it manages the execution of the application logic on allocated containers.

The YARN architecture
In the previous topic, we discussed the YARN components. Here we'll discuss the high-level architecture of YARN and look at how the components interact with each other.
The ResourceManager service runs on the master node of the cluster. A YARN client submits an application to the ResourceManager. An application can be a single MapReduce job, a directed acyclic graph of jobs, a java application, or any shell script. The client also defines an ApplicationMaster and a command to start the ApplicationMaster on a node.

The ApplicationManager service of resource manager will validate and accept the application request from the client. The scheduler service of resource manager will allocate a container for the ApplicationMaster on a node and the NodeManager service on that node will use the command to start the ApplicationMaster service. Each YARN application has a special container called ApplicationMaster. The ApplicationMaster container is the first container of an application.

The ApplicationMaster requests resources from the ResourceManager. The RequestRequest will have the location of the node, memory, and CPU cores required. The ResourceManager will allocate the resources as containers on a set of nodes. The ApplicationMaster will connect to the NodeManager services and request NodeManager to start containers. The ApplicationMaster manages the execution of the containers and will notify the ResourceManager once the application execution is over. Application execution and progress monitoring is the responsibility of ApplicationMaster rather than ResourceManager.

The NodeManager service runs on each slave of the YARN cluster. It is responsible for running application's containers. The resources specified for a container are taken from the NodeManager resources. Each NodeManager periodically updates ResourceManager for the set of available resources. The ResourceManager scheduler service uses this resource matrix to allocate new containers to ApplicationMaster or to start execution of a new application.

How YARN satisfies big data needs
We talked about the MapReduce v1 framework and some limitations of the framework. Let's now discuss how YARN solves these issues:
• **Scalability and higher cluster utilization:** Scalability is the ability of a software or product to implement well under an expanding workload. In YARN, the responsibility of resource management and job scheduling/monitoring is divided into separate daemons, allowing YARN daemons to scale the cluster without degrading the performance of the cluster.

With a flexible and generic resource model in YARN, the scheduler handles an overall resource profile for each type of application. This structure makes the communication and storage of resource requests efficient for the scheduler resulting in higher cluster utilization.

• **High availability for components:** Fault tolerance is a core design principle for any multitenancy platform such as YARN. This responsibility is delegated to ResourceManager and ApplicationMaster. The application specific framework, ApplicationMaster, handles the failure of a container. The ResourceManager handles the failure of NodeManager and ApplicationMaster.

• **Flexible resource model:** In MapReduce v1, resources are defined as the number of map and reduce task slots available for the execution of a job. Every resource request cannot be mapped as map/reduce slots. In YARN, a resource-request is defined in terms of memory, CPU, locality, and so on. It results in a generic definition for a resource request by an application. The NodeManager node is the worker node and its capability is calculated based on the installed memory and cores of the CPU.
Starting with YARN Basics

- **Multiple data processing algorithms:** The MapReduce framework is bounded to batch processing only. YARN is developed with a need to perform a wide variety of data processing over the data stored over Hadoop HDFS. YARN is a framework for generic resource management and allows users to execute multiple data processing algorithms over the data.

- **Log aggregation and resource localization:** As discussed earlier, accessing and managing user logs is difficult in the Hadoop 1.x framework. To manage user logs, YARN introduced a concept of log aggregation. In YARN, once the application is finished, the NodeManager service aggregates the user logs related to an application and these aggregated logs are written out to a single log file in HDFS. To access the logs, users can use either the YARN command-line options, YARN web interface, or can fetch directly from HDFS.

A container might require external resources such as jars, files, or scripts on a local file system. These are made available to containers before they are started. An ApplicationMaster defines a list of resources that are required to run the containers. For efficient disk utilization and access security, the NodeManager ensures the availability of specified resources and their deletion after use.
Projects powered by YARN

Efficient and reliable resource management is a basic need of a distributed application framework. YARN provides a generic resource management framework to support data analysis through multiple data processing algorithms. There are a lot of projects that have started using YARN for resource management. We've listed a few of these projects here and discussed how YARN integration solves their business requirements:

- **Apache Giraph**: Giraph is a framework for offline batch processing of semistructured graph data stored using Hadoop. With the Hadoop 1.x version, Giraph had no control over the scheduling policies, heap memory of the mappers, and locality awareness for the running job. Also, defining a Giraph job on the basis of mappers / reducers slots was a bottleneck. YARN's flexible resource allocation model, locality awareness principle, and application master framework ease the Giraph's job management and resource allocation to tasks.

- **Apache Spark**: Spark enables iterative data processing and machine learning algorithms to perform analysis over data available through HDFS, HBase, or other storage systems. Spark uses YARN's resource management capabilities and framework to submit the DAG of a job. The spark user can focus more on data analytics' use cases rather than how spark is integrated with Hadoop or how jobs are executed.

Some other projects powered by YARN are as follows:

- **MapReduce**: https://issues.apache.org/jira/browse/MAPREDUCE-279
- **Giraph**: https://issues.apache.org/jira/browse/GIRAPH-13
- **Spark**: http://spark.apache.org/
- **OpenMPI**: https://issues.apache.org/jira/browse/MAPREDUCE-2911
- **HAMA**: https://issues.apache.org/jira/browse/HAMA-431
- **HBase**: https://issues.apache.org/jira/browse/HBASE-4329
- **Storm**: http://hortonworks.com/labs/storm/

A page on Hadoop wiki lists a number of projects/applications that are migrating to or using YARN as their resource management tool. You can see this at http://wiki.apache.org/hadoop/PoweredByYarn.
Summary
It is time to summarize the learning from this chapter and let you know what's to come in the next chapter. In this chapter, you learnt about the MapReduce v1 framework and its shortcomings. The chapter also covered an introduction to YARN, its components, architecture, and different projects powered by YARN. It also explained how YARN solves big data needs. In the next chapter, you will create single as well as multiple node Hadoop-YARN clusters and begin with your first step to YARN.
Where to buy this book

You can buy Learning YARN from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.