IPASJ International Journal of Computer Science (IIJCS)
Web Site: http://www.ipasj.org/IIJCS/IIJCS.htm
A Publisher for Research Motivation ....
Email: editoriijcs@ipasj.org
Volume 3, Issue 9, September 2015
ISSN 2321-5992

Hadoop2 Yarn

Asso.Prof. Ashish Sharma [#1] **Snehlata Vyas** [*2]

[#] Computer Science Jodhpur National University
,Jodhpur, Rajasthan,India

[#]ComputerScience,Mahila PG Mahavidhyalaya,JNVU
Jodhpur, Rajasthan,India

ABSTRACT

Apache Hadoop 2 (Hadoop 2.0) is the second iteration of the Hadoop framework for distributed data processing Hadoop 2 adds support for running non-batch applications through the introduction of YARN, a redesigned cluster resource manager that eliminates Hadoop's sole reliance on the MapReduce programming model. Short for Yet Another Resource Negotiator, YARN puts resource management and job scheduling functions in a separate layer beneath the data processing one, enabling Hadoop 2 to run a variety of applications. Overall, the changes made in Hadoop 2 position the framework for wider use in big data analytics and other enterprise applications. For example, it is now possible to run event processing as well as streaming, real-time and operational applications. HDFS federation brings important measures of scalability and reliability to Hadoop. YARN, the other major advance in Hadoop 2, brings significant performance improvements for some applications, supports additional processing models, and implements a more flexible execution engine. YARN is a resource manager that was created by separating the processing engine and resource management capabilities of MapReduce as it was implemented in Hadoop 1.YARN is often called the operating system of Hadoop because it is responsible for managing and monitoring workloads, maintaining a multi-tenant environment, implementing security controls, and managing high availability features of Hadoop

Keywords: Hadoop2.0,Yarn,MapReduce,Big Data,Cluster,High Availability, ResourceManager, NodeManager

1.INTRODUCTION

YARN is one of the key features in the second-generation Hadoop 2 version of the Apache Software Foundation's open source distributed processing framework. Originally described by Apache as a redesigned resource manager, YARN is now characterized as a large-scale, distributed operating system for big data applications.

Hadoop is a free, Java-based programming framework that supports the processing of large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation. Hadoop makes it possible to run applications on systems with thousands of nodes involving thousands of terabytes. Its distributed file system facilitates rapid data transfer rates among nodes and allows the system to continue operating uninterrupted in case of a node failure. This approach lowers the risk of catastrophic system failure, even if a significant number of nodes become inoperative.

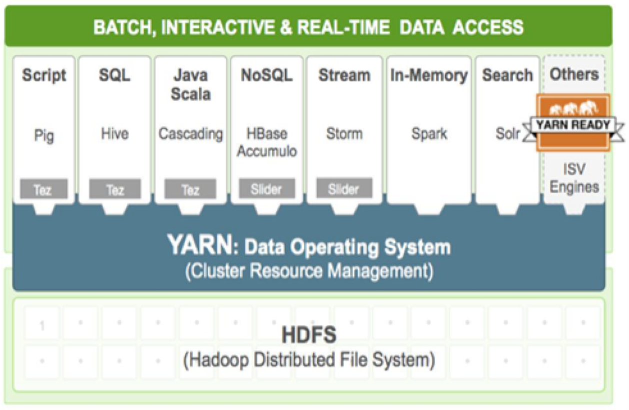
Hadoop is a software framework in which an application is broken down into numerous small parts. Any of these parts (also called fragments or blocks) can be run on any node in the cluster. The current Apache Hadoop ecosystem consists of the Hadoop kernel, MapReduce, the Hadoop distributed file system (HDFS) and a number of related projects such as Apache Hive, HBase and Zookeeper data is an evolving term that describes any voluminous amount of structured, semi-structured and unstructured data that has the potential to be mined for information. Big data refer to petabytes and exabytes of data.

YARN is a resource manager that was created by separating the processing engine and resource management capabilities of MapReduce as it was implemented in Hadoop 1. YARN is often called the operating system of Hadoop because it is responsible for managing and monitoring workloads, maintaining a multi-tenant environment, implementing security controls, and managing high availability features of Hadoop.

Like an operating system on a server, YARN is designed to allow multiple, diverse user applications to run on a multi-tenant platform. In Hadoop 1, users had the option of writing MapReduce programs in Java, in Python, Ruby or other scripting languages using streaming, or using Pig, a data transformation language. Regardless of which method was used, all model to run. fundamentally relied on the MapReduce processing.

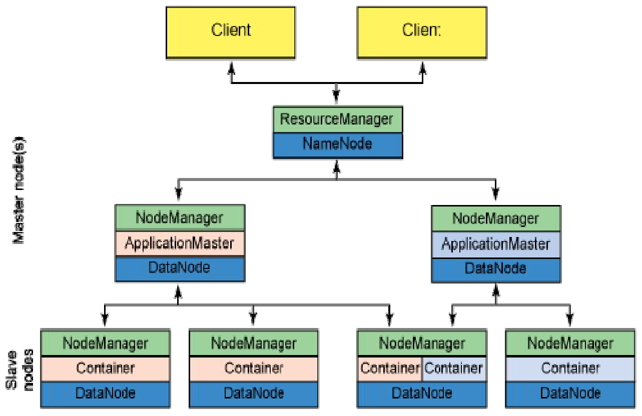
YARN supports multiple processing models in addition to MapReduce. One of the most significant benefits of this is that we are no longer limited to working the often I/O intensive, high latency MapReduce framework.

Hadoop 2 also includes new features designed to improve system availability and scalability. For example, it introduced an Hadoop Distributed File System (HDFS) high-availability (HA) feature that brings a new NameNode architecture to Hadoop. The Hadoop 2 high-availability scheme allows users to configure clusters with redundant NameNodes, removing the chance that a lone NameNode will become a single point of failure (SPoF) within a cluster. Meanwhile, a new HDFS federation capability lets clusters be built out horizontally with multiple NameNodes that work independently but share a common data storage pool, offering better compute scaling as compared to Apache Hadoop 1.x.



.

2. ARCHITECTURE OF HADOOP2(YARN)



The new architecture for YARN

At the root of a YARN hierarchy is the ResourceManager. This entity governs an entire cluster and manages the assignment of applications to underlying compute resources. The ResourceManager orchestrates the division of resources (compute, memory, bandwidth, etc.) to underlying NodeManagers (YARN's per-node agent). The ResourceManager also works with ApplicationMasters to allocate resources and work with the NodeManagers to start and monitor their underlying application. In this context, the ApplicationMaster has taken some of the role of the prior TaskTracker, and the ResourceManager has taken the role of the JobTracker.

An ApplicationMaster manages each instance of an application that runs within YARN. The ApplicationMaster is responsible for negotiating resources from the ResourceManager and, through the NodeManager, monitoring the execution and resource consumption of containers (resource allocations of CPU, memory, etc.)..

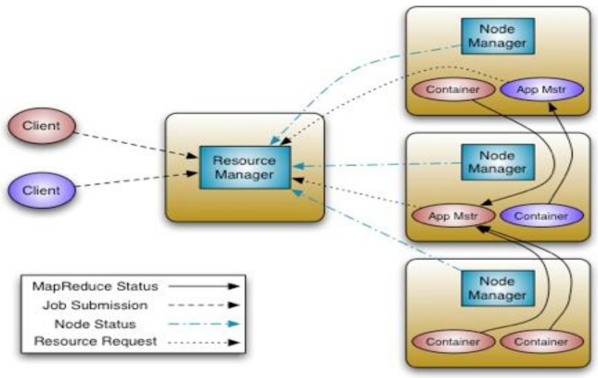
The NodeManager manages each node within a YARN cluster. The NodeManager provides per-node services within the cluster, from overseeing the management of a container over its life cycle to monitoring resources and tracking the health of its node. Unlike MRv1, which managed execution of map and reduce tasks via slots, the NodeManager manages abstract containers, which represent per-node resources available for a particular application. YARN continues to use the HDFS layer, with its master NameNode for metadata services and DataNode for replicated storage services across a cluster.

Use of a YARN cluster begins with a request from a client consisting of an application. The ResourceManager negotiates the necessary resources for a container and launches an ApplicationMaster to represent the submitted application. Using a resource-request protocol, the ApplicationMaster negotiates resource containers for the application at each node. Upon execution of the application, the ApplicationMaster monitors the container until completion. When the application is complete, the ApplicationMaster unregisters its container with the ResourceManager, and the cycle is complete.

The YARN architecture allow a new ResourceManager to manage resource usage across applications, with ApplicationMasters taking the responsibility of managing the execution of jobs. This change removes a bottleneck and also improves the ability to scale Hadoop clusters to much larger configurations than previously possible. In addition, beyond traditional MapReduce, YARN permits simultaneous execution of a variety of programming models, including graph

processing, iterative processing, machine learning, and general cluster computing, using standard communication schemes like the Message Passing Interface.
The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker i.e. resourcemanagement and job scheduling/monitoring, into separate daemons: a global ResourceManager and per-application ApplicationMaster (AM).

The ResourceManager and per-node slave, the NodeManager (NM), form the new, and generic, **system** for managing applications in a distributed manner.



Application execution consists of the following steps:
- Application submission.
- Bootstrapping the ApplicationMaster instance for the application.
- Application execution managed by the ApplicationMaster instance.

An application execution sequence:

1. A client program submits the application, including the necessary specifications to launch the application-specific ApplicationMaster itself.

2. The ResourceManager assumes the responsibility to negotiate a specified container in which to start the ApplicationMaster and then launches the ApplicationMaster.

3. The ApplicationMaster, on boot-up, registers with the ResourceManager – the registration allows the client program to query the ResourceManager for details, which allow it to directly communicate with its own ApplicationMaster.

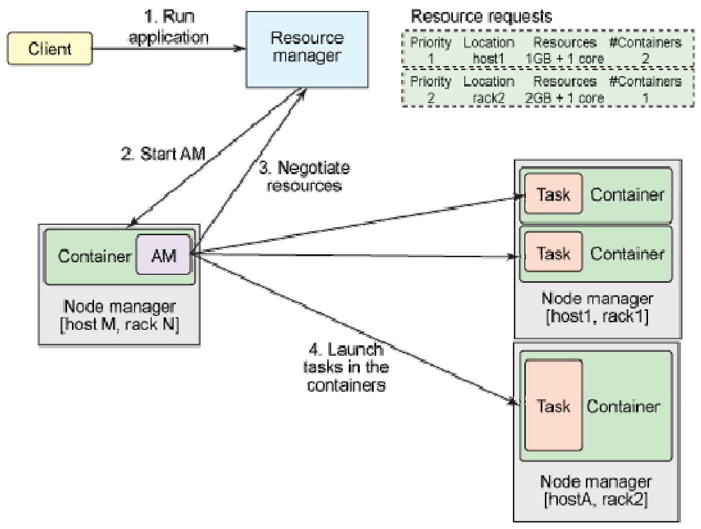
4. During normal operation the ApplicationMaster negotiates appropriate resource containers via the resource-request protocol.

5. On successful container allocations, the ApplicationMaster launches the container by providing the container launch specification to the NodeManager. The launch specification, typically, includes the necessary information to allow the container to communicate with the ApplicationMaster itself.

6. The application code executing within the container then provides necessary information (progress, status etc.) to its ApplicationMaster via an application-specific protocol.

7. During the application execution, the client that submitted the program communicates directly with the ApplicationMaster to get status, progress updates etc. via an application-specific protocol.

8. Once the application is complete, and all necessary work has been finished, the ApplicationMaster deregisters with the ResourceManager and shuts down, allowing its own container to be repurposed.


Client
1. Run application
Resource manager
Resource requests
Priority Location Resources #Containers
1 host1 1GB + 1 core 2
Priority Location Resources #Containers
2 rack2 2GB + 1 core 1
2. Start AM
3. Negotiate resources
Container
AM
Node manager [host M, rack N]
Task
Container
Task
Container
Node manager [host1, rack1]
4. Launch tasks in the containers
Task
Container
Node manager [hostA, rack2]

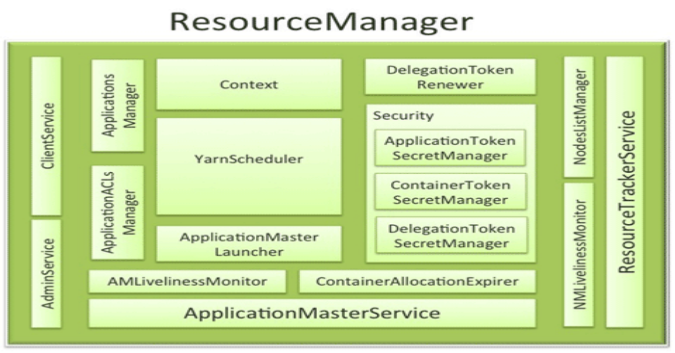
Application submission in YARN

1) Apache Hadoop YARN – ResourceManager

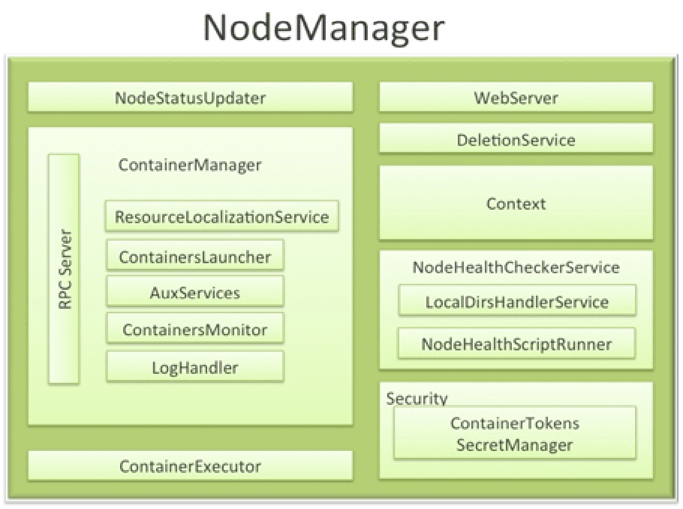
ResourceManager (RM**)** is the master that arbitrates all the available cluster resources and thus helps manage the distributed applications running on the YARN system. It works together with the per-node NodeManagers (NMs**)** and the per-application ApplicationMasters (AMs)**.**

1. NodeManagers take instructions from the ResourceManager and manage resources available on a single node.

2. ApplicationMasters are responsible for negotiating resources with the ResourceManager and for working with the NodeManagers to start the containers.



2)NodeManager

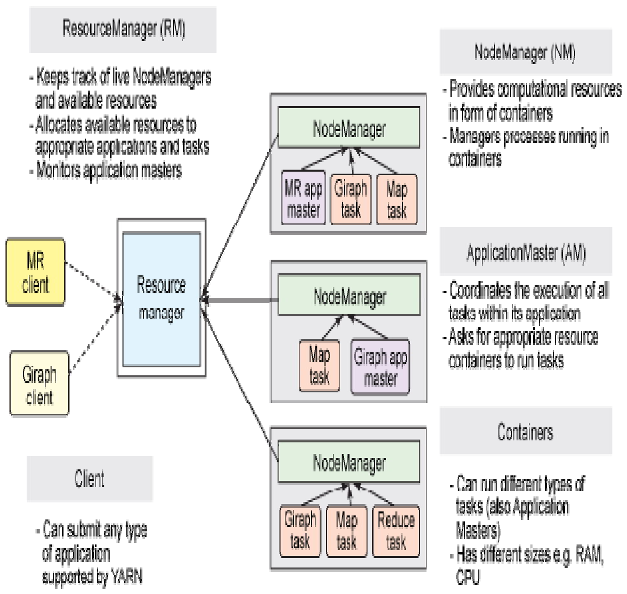


NodeStatusUpdater
On startup, this component registers with the RM and sends information about the resources available on the nodes. Subsequent NM-RM communication is to provide updates on container statuses – new containers running on the node, completed containers, etc.
In addition the RM may signal the NodeStatusUpdater to potentially kill already running containers.

ContainerManager

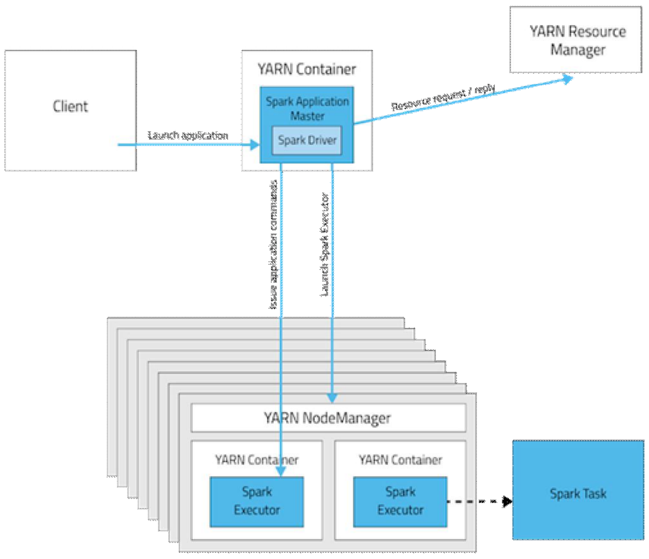
This is the core of the NodeManager. It is composed of the following sub-components, each of which performs a subset of the functionality that is needed to manage containers running on the node.

In YARN, the NodeManager is primarily limited to managing abstract containers i.e. only processes corresponding to a container and not concerning itself with per-application state management like MapReduce tasks. It also does away with the notion of named slots like map and reduce slots. Because of this clear separation of responsibilities coupled with the modular architecture described above, NM can scale much more easily and its code is much more maintainable.


ResourceManager (RM)
- Keeps track of live NodeManagers and available resources
- Allocates available resources to appropriate applications and tasks
- Monitors application masters
NodeManager (NM)
- Provides computational resources in form of containers
- Managers processes running in containers
NodeManager
MR app master
Giraph task
Map task
MR client
Resource manager
Giraph client
NodeManager
Map task
Giraph app master
ApplicationMaster (AM)
- Coordinates the execution of all tasks within its application
- Asks for appropriate resource containers to run tasks
Containers
NodeManager
Giraph task
Map task
Reduce task
Client
- Can submit any type of application supported by YARN
- Can run different types of tasks (also Application Masters)
- Has different sizes e.g. RAM, CPU
Architecture of YARN

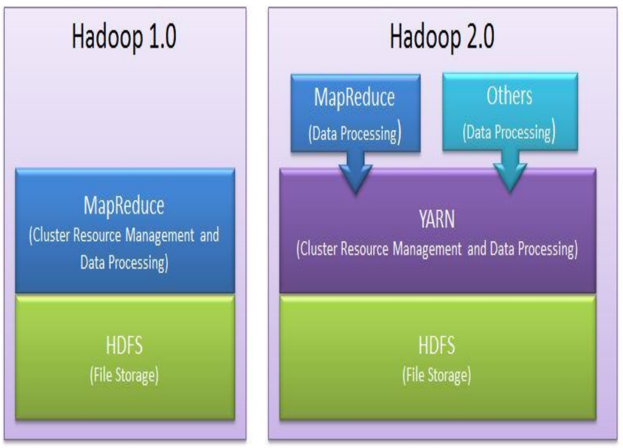
In the YARN architecture, a global ResourceManager runs as a master daemon, usually on a dedicated machine, that arbitrates the available cluster resources among various competing applications. The ResourceManager tracks how many live nodes and resources are available on the cluster and coordinates what applications submitted by users should get these resources and when. The ResourceManager is the single process that has this information so it can make its allocation (or rather, scheduling) decisions in a shared, secure, and multi-tenant manner (for instance, according to an application priority, a queue capacity, ACLs, data locality, etc.).
When a user submits an application, an instance of a lightweight process called the ApplicationMaster is started to coordinate the execution of all tasks within the application. This includes monitoring tasks, restarting failed tasks, speculatively running slow tasks, and calculating total values of application counters. These responsibilities were previously assigned to the single JobTracker for all jobs. The ApplicationMaster and tasks that belong to its application run in resource containers controlled by the NodeManagers.
The NodeManager is a more generic and efficient version of the TaskTracker. Instead of having a fixed number of map and reduce slots, the NodeManager has a number of dynamically created resource containers. The size of a container depends upon the amount of resources it contains, such as memory, CPU, disk, and network IO. Currently, only memory and CPU are supported. groups might be used to control disk and network IO in the future. The number of containers on a node is a product of configuration parameters and the total amount of node resources (such as total CPUs and total memory) outside the resources dedicated to the slave daemons and the OS.
Interestingly, the ApplicationMaster can run any type of task inside a container. For example, the MapReduce ApplicationMaster requests a container to launch a map or a reduce task, while the Giraph ApplicationMaster requests a container to run a Giraph task. You can also implement a custom ApplicationMaster that runs specific tasks and, in this way, invent a shiny new distributed application framework that changes the big data world. I encourage you to read about Apache Twill, which aims to make it easy to write distributed applications sitting on top of YARN.

In YARN, MapReduce is simply degraded to a role of a distributed application (but still a very popular and useful one) and is now called MRv2. MRv2 is simply the re-implementation of the classical MapReduce engine, now called MRv1, that runs on top of YARN.
A Hadoop cluster is a special type of computational cluster designed specifically for storing and analyzing huge amounts of unstructured data in a distributed computing environment.
Such clusters run Hadoop's open source distributed processing software on low-cost commodity computers. Typically one machine in the cluster is designated as the namenode and another machine the as jobtracker; these are the masters. The rest of the machines in the cluster act as both datanode and tasktracker; these are the slaves. Hadoop clusters are often referred to as "shared nothing" systems because the only thing that is shared between nodes is the network that connects them.
Hadoop clusters are known for boosting the speed of data analysis applications. They also are highly scalable: If a cluster's processing power is overwhelmed by growing volumes of data, additional cluster nodes can be added to increase throughput. Hadoop clusters also are highly resistant to failure because each piece of data is copied onto other cluster nodes, which ensures that the data is not lost if one node fails. A Hadoop cluster is a special type of cluster that is specifically designed for storing and analyzing huge amounts of unstructured data. A Hadoop cluster is essentially a computational cluster that distributes the data analysis workload across multiple cluster nodes that work to process the data in parallel.
As of early 2013, Facebook was recognized as having the largest Hadoop cluster in the world. Other prominent users include Google, Yahoo and IBM.
A Hadoop cluster is a special type of cluster that is specifically designed for storing and analyzing huge amounts of unstructured data. A Hadoop cluster is essentially a computational cluster that distributes the data analysis workload across multiple cluster nodes that work to process the data in parallel.



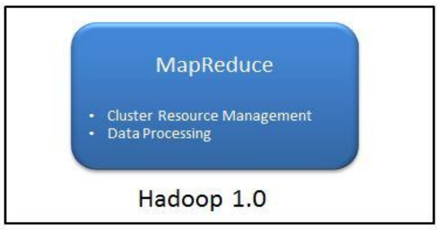
3. HOW YARN OVERCOMES MAPREDUCE LIMITATIONS IN HADOOP 2.0

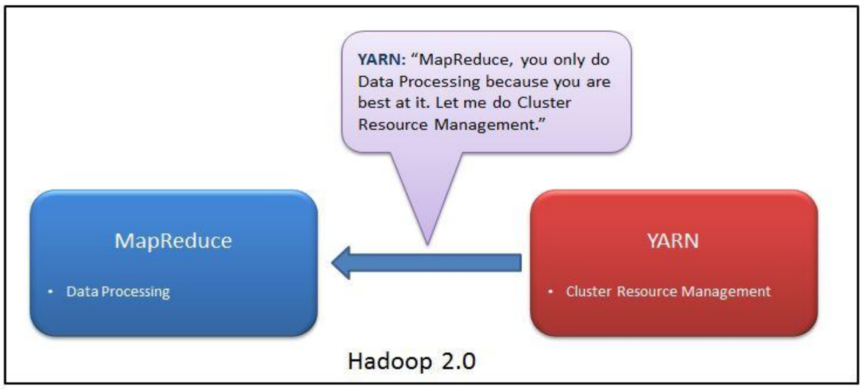
Introduction of new YARN layer in Hadoop 2.0:



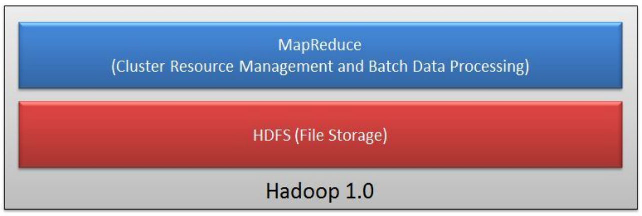
IPASJ International Journal of Computer Science (IIJCS)
Web Site: http://www.ipasj.org/IIJCS/IIJCS.htm
A Publisher for Research Motivation .... Email: editoriijcs@ipasj.org
Volume 3, Issue 9, September 2015 ISSN 2321-5992

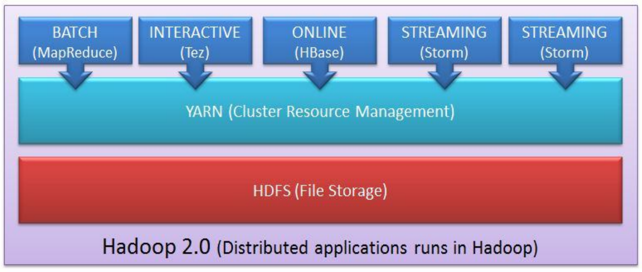
As shown, in Hadoop 2.0 a new layer has been introduced between HDFS and MapReduce.
This is YARN framework which is responsible for doing Cluster Resource Management.
Cluster Resource Management:
Cluster resource management means managing the resources of the Hadoop Clusters. And by resources we mean Memory, CPU etc.
YARN took over this task of cluster management from MapReduce and MapReduce is streamlined to perform Data Processing only in which it is best.




YARN: "MapReduce, you only do Data Processing because you are best at it. Let me do Cluster Resource Management."
MapReduce
• Data Processing
YARN
• Cluster Resource Management
Hadoop 2.0

Why YARN was needed
Before we understand the need of YARN, we should understand how cluster resource management was done in Hadoop 1.0 and what the problem in that approach was.
In MapReduce framework, MapReduce job (MapReduce application) is divided between number of tasks called mappers and reducers. Each task runs on one of the machine (DataNode) of the cluster, and each machine has a limited number of predefined slots (map slot, reduce slot) for running tasks concurrently.
Here, JobTracker is responsible for both managing the cluster's resources and driving the execution of the MapReduce job. It reserves and schedules slots for all tasks, configures, runs and monitors each task, and if a task fails, it allocates a new slot and reattempts the task. After a task finishes, the job tracker cleans up temporary resources and releases the task's slot to make it available for other jobs.
Problems with this approach in Hadoop 1.0:
It limits scalability
Availability Issue: In Hadoop 1.0, JobTracker is single Point of availability. This means if JobTracker fails, all jobs must restart.
Problem with Resource Utilization: In Hadoop 1.0, there is concept of predefined number of map slots and reduce slots for each TaskTrackers resources to be used as Mapper slots.
Limitation in running non-MapReduce Application: In Hadoop 1.0, Job tracker was tightly integrated with MapReduce and only supporting application that obeys MapReduce programming framework can run on Hadoop.
Let's try to understand point 4 in more detail.
Problem in performing real-time analysis: MapReduce is batch driven. What if I want to do perform real time analysis instead of batch-processing (where results is available after several hours).
Problem in running Message-Passing approach: It is a stateful process that runs on each node of a distributed network. The processes communicate with each other by sending messages, and alter their state based on the messages they receive. This is not possible in MapReduce.
Problem in running Ad-hoc query: Many users like to query their big data using SQL. Apache Hive can execute a SQL query as a series of MapReduce jobs, but it has shortcomings in terms of performance.
Hadoop 2.0 solves all these problem with YARN:

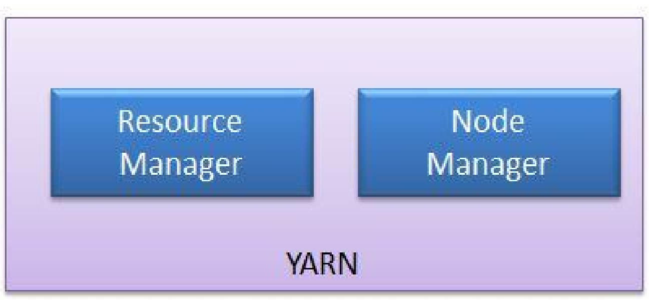




YARN took over the task of cluster management from MapReduce and MapReduce is streamlined to perform Data Processingonly in which it is best.
YARN has central resource manager component which manages resources and allocates the resources to the application. Multiple applications can run on Hadoop via YARN and all application could share common resource management.

4.ADVANTAGE OF YARN

1 .Yarn does efficient utilization of the resource.
There are no more fixed map-reduce slots. YARN provides central resource manager. With YARN, you can now run multiple applications in Hadoop, all sharing a common resource.
2.Yarn can even run application that do not follow MapReduce model.
YARN decouples MapReduce's resource management and scheduling capabilities from the data processing component, enabling Hadoop to support more varied processing approaches and a broader array of applications. For example, Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs. This also streamlines MapReduce to do what is does best - process data.
Few Important Notes about YARN:
3.YARN is backward compatible.
This means that existing MapReduce job can run on Hadoop 2.0 without any change.
4.No more JobTracker and TaskTracker needed in Hadoop 2.0
JobTracker and TaskTracker has totally disappeared. YARN splits the two major functionalities of the JobTracker i.e. resource management and job scheduling/monitoring into 2 separate daemons (components).
Resource Manager
Node Manager(node specific)
Central Resource Manager and node specific Node Manager together constitutes YARN.



MapReduce: Difference between MR1 and MR2:

Earlier version of map- reduce framework in Hadoop 1.0 is called as **MR1**. The new version of MapReduce is known as **MR2**.
No more JobTracker and TaskTracker needed in Hadoop 2. With the introduction of YARN in Hadoop2, the term JobTracker and TaskTracker disappeared. MapReduce is now streamlined to perform processing data. The new model is more isolated and scalable as compared to the earlier MR1 system. MR2 is one kind of distributed application that run MapReduce framework on top of YARN. MapReduce perform data processing via YARN. Other tools can also perform data processing via YARN. Hence Yarn execution model is more generic than earlier MapReduce model. MR1 was not able to do so. It would only run MapReduce applications.

5. CONCLUSION

We have entered in the world of Big Data. The paper describes the concept of Apache Hadoop 2 (Hadoop 2.0)Yarn, the second iteration of the Hadoop framework for distributed data processing Data along. The paper also focuses the limitation of hadoop and Cluster Resource Management in Hadoop2. These technical challenges must be addressed for efficient and fast processing of Big Data. The paper describes Architecture of Hadoop2 which is an open source software used for processing of Big Data and overcome the limitations of Hadoop1 and Advantage of YARN.

REFERENCES

[1] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein "Online Aggregation and Continuous Query support in MapReduce" SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA. Copyright 2010 ACM 978-1-4503-0032-2/10/06
[2] Jonathan Paul Olmsted "Scaling at Scale: Ideal Point Estimation with 'Big-Data" Princeton Institute for Computational Science and Engineering 2014.
[3] Jonathan Stuart Ward and Adam Barker "Undefined By Data: A Survey of Big Data Definitions" Stamford, CT: Gartner, 2012.
[4] Chen He Ying Lu David Swanson "Matchmaking: A New MapReduce Scheduling" in 10th IEEE International Conference on Computer and Information Technology (CIT'10), pp. 2736–2743, 2010
[5] Ahmed Eldawy, Mohamed F. Mokbel "A Demonstration of SpatialHadoop:An Efficient MapReduce Framework for Spatial Data" Proceedings of the VLDB Endowment, Vol. 6, No. 12 Copyright 2013 VLDB Endowment 21508097/13/10.
[6] Hadoop http://hadoop.apache.org
http://sortbenchmark.org/YahooHadoop.pdf
[7] Hadoop. http://hadoop.apache.org, 2009. [4] HDFS (hadoop distributed file system) architecture. http://hadoop. apache.org/common/docs/current/hdfs design.html, 2009.
[8] R. Abbott and H. Garcia-Molina. Scheduling I/O requests with deadlines: A performance evaluation. In Proceedings of the 11th Real-Time Systems Symposium, pages 113–124, Dec 1990.
[9] G. Candea, N. Polyzotis, and R. Vingralek. A scalable, predictable join operator for highly concurrent data warehouses. In 35th International Conference on Very Large Data Bases (VLDB), 2009.
[10] R.-I. Chang, W.-K. Shih, and R.-C. Chang. Deadline-modificationscan with maximum-scannable-groups for multimedia real-time disk scheduling. In RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium, page 40, Washington, DC, , USA, 1998. IEEE Computer Society
[11] S. Iyer and P. Druschel. Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O. In SOSP '01: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, pages 117–130, New York, NY, USA, 2001. ACM.
[12] S. C. John, J. A. Stankovic, J. F. Kurose, and D. Towsley. Performance evaluation of two new disk scheduling algorithms for real-time systems. Journal of Real-Time Systems, 3:307–336, 1991.
[13] S. Kashyap, B. Olszewski, and R. Hendrickson. Improving database performance with AIX concurrent I/O: A case study with oracle9i database on AIX 5L version 5.2. Technical report, IBM, 2003.
[14] B. Ndiaye, X. Nie, U. Pathak, and M. Susairaj. A quantitative comparison between raw devices and file systems for implementing oracle databases. Technical report, Oracle / Hewlett-Packard, 2004.
[15] O. O'Malley and A. C. Murthy. Winning a 60 second dash with a yellow elephant. Technical report, Yahoo!, 2009.
[16] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In SIGMOD '09: Proceedings of the 35th SIGMOD International Conference on Management of Data, pages 165–178, New York, NY, USA, 2009. ACM.