International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3. 358

# An Enhanced Approach for Resource Management Optimization in Hadoop

**R. Sandeep Raj[1], G. Prabhakar Raju [2]**

[1]MTech Student, Department of CSE, Anurag Group of Institutions, India

[2]Associate Professor, Department of CSE, Anurag Group of Institutions, India

**Abstract:** ***Many tools and frameworks have been developed to process data on distributed data centers. MapReduce[3] most prominent among such frameworks has emerged as a popular distributed data processing model for processing vast amount of data in parallel on large clusters of commodity machines. The JobTracker in MapReduce framework is responsible for both managing the cluster's resources and executing the MapReduce jobs, a constraint that limits scalability, resource utilization. YARN [2] the next-generation execution layer for Hadoop splits processing and resource management capabilities of JobTracker into separate entities and eliminates the dependency of Hadoop on MapReduce. This new model is more isolated and scalable compared to MapReduce, providing improved features and functionality. This paper discusses the design of YARN and significant advantages over traditional MapReduce.***

**Keywords:** Big Data, Hadoop, MapReduce, YARN, Scalability

## 1. Introduction

Hadoop [1] is an open source framework for processing vast amount of data on large clusters of commodity hardware in parallel. Hadoop is accessible, simple, scalable and robust. Hadoop is well known for its processing framework, Mapreduce and its distributed file system HDFS[4]. HDFS is designed to hold large amount of data, and provide access to this data to many clients distributed across a network. HDFS has two categories of nodes: Namenode and Datanodes. The Namenode maintain metadata of all the files and the Datanodes are the responsible for storing the actual blocks of all the files in the filesystem. MapReduce is programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. Hadoop is an open source implementation of MapReduce.

Hadoop is an open source framework for writing and running distributed applications that process large amounts of data. Distributed computing is a wide and varied field, but the key distinctions of Hadoop are that it is;

- Accessible—Hadoop runs on large clusters of commodity machines or on cloud computing services such as Amazon's Elastic Compute Cloud (EC2).
- Robust—Because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.
- Scalable—Hadoop scales linearly to handle larger data by adding more nodes to the cluster.
- Simple—Hadoop allows users to quickly write efficient parallel code.
- Hadoop's accessibility and simplicity give it an edge over writing and running large distributed programs. Even students can quickly and cheaply create their own Hadoop cluster. On the other hand, its robustness and scalability make it suitable for even the most demanding jobs at Yahoo and Facebook. These features make Hadoop popular in both academia and industry. Hadoop's main processing engine is MapReduce, which is currently one of the most popular big-data processing frameworks available. Numerous practical problems ranging from log analysis, to data sorting, to text processing, to pattern-based search, to graph processing, to machine learning, and much more have been solved using MapReduce.

### 1. 1 Literature Survey

Several studies were published on tuning the performance of MapReduce. Many have recognized the limitations of classic MapReduce programming model. Mesos[13] is a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. Sharing improves cluster utilization and avoids per-framework data replication. Mesos shares resources in a fine-grained manner, allowing frameworks to achieve data locality by taking turns reading data stored on each machine. To support the sophisticated schedulers of today's frameworks, Mesos introduces a distributed two-level scheduling mechanism called resource offers. Mesos decides how many resources to offer each framework, while frameworks decide which resources to accept and which computations to run on them. Mesos can achieve near-optimal data locality when sharing the cluster among diverse frameworks, can scale to 50,000 nodes, and is resilient to failures.

Omega[14], Google's next-generation cluster management platform is focused on a cluster scheduling architecture that uses parallelism, shared state, and optimistic concurrency control. Omega model uses both lightweight simulations with synthetic workloads, and high-fidelity, trace-based simulations of production workloads at Google,and the evaluation shows that optimistic concurrency over shared state is a viable, attractive approach to cluster scheduling.

The Cosmos[15] Storage System is an append-only file system that reliably stores petabytes of data. The system is optimized for large sequential I/O. All writes are append-only and concurrent writers are serialized by the system. Data is distributed and replicated for fault tolerance and compressed to save storage and increase I/O throughput. In Cosmos, applications are programmed against the execution engine that provides a higher-level programming interface and a runtime system that automatically handles the details of optimization, fault tolerance, data partitioning, resource management, and parallelism.

### 1. 2 Mapreduce Processing

According to [3], MapReduce is a programming model for processing large-scale datasets in computer clusters. The MapReduce programming model consists of two functions, map () and reduce (). Users can implement their own processing logic by specifying a customized map () and reduce () function. The map () function takes an input key/value pair and produces a list of intermediate key/value pairs. The MapReduce runtime system groups together all intermediate pairs based on the intermediate keys and passes them to reduce () function for producing the final results. The signatures of map () and reduce () are as follows;

map (k1,v1) → list(k2,v2)
reduce (k2,list(v2)) → list(v2)

A MapReduce cluster employs a master-slave architecture where one master node manages a number of slave nodes. In the Hadoop, the master node is called JobTracker and the slave node is called TaskTracker. Hadoop launches a MapReduce job by first splitting the input dataset into even-sized data blocks. Each data block is then scheduled to one TaskTracker node and is processed by a map task. The task assignment process is implemented as a heartbeat protocol. The TaskTracker node notifies the JobTracker when it is idle. The scheduler then assigns new tasks to it. The scheduler takes data locality into account when it disseminates data blocks. It always tries to assign a local data block to a TaskTracker. If the attempt fails, the scheduler will assign a rack-local or random data block to the TaskTracker instead. When map() functions complete, the runtime system groups all intermediate pairs and launches a set of reduce tasks to produce the final results. .

## 2. Motivation

Hadoop's MapReduce is an easy-to-use distributed data processing framework. The analysis reveals a number of limitations, including;

a) Limitation on scalability
b) Inefficient resource utilization and
c) Lack of support for non-Mapreduce applications
d) Availabilty issue

### 2. 1 Limitation on Scalability

The classic MapReduce is composed of a master node, JobTracker and a number of slave nodes, TaskTrackers. The JobTracker is in charge of two distinct, complex responsibilities.

- Managing the computational resources in the cluster, which involves maintaining the list of live nodes, the list of available and occupied map and reduce slots, and allocating the available slots to appropriate jobs and tasks according to selected scheduling policy [5].
- Coordination of all tasks running on a cluster, which involves instructing TaskTrackers to start map and reduce tasks, monitoring the execution of the tasks, restarting failed tasks, speculatively running slow tasks, calculating total values of job counters, and more.

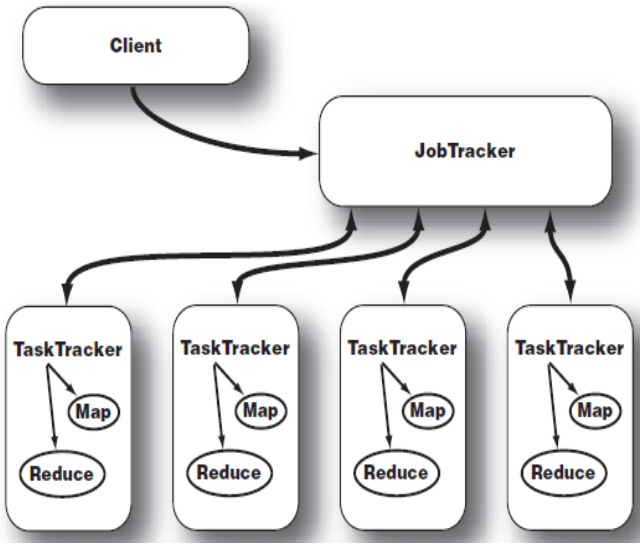


**Figure1:** Running MapReduce on MR1

These large number of responsibilities on a single process caused scalability issues, especially on clusters where jobtracker had to monitor a large number of tasktrackers, jobs submitted and the execution of map,reduce tasks.

### 2. 2 Inefficient Resource Utilization

Hadoop clusters never use the computational resources efficiently. The computational resources are split into fixed number of concurrent map slots and reduce slots[8] on a node by the administrator in MapReduce framework. Inefficient resource utilization occurs because mapper slots might be full while the reducer slots are totally empty. The Datanodes that run reduce slots may be idle even though there is immediate requirement for compute resources to be used by mapper slots.

### 2. 3 Lack of Support For Non-Mapredue Workloads

The JobTracker is tightly coupled with the Mapreduce constraining Hadoop to run only Mapreduce based applications. Mapreduce is I/O intensive [9] and is not suitable for running memory intensive programming paradigms such as graph processing, machine learning.

Though MapReduce supports different use cases, it is not suitable for large scale computation and is not flexible to integrate a wide range of emerging projects.

### 2.4 Addressing the availability issue

#### 2.4.1 Tasktracker Failure

If a tasktracker fails by crashing, the JobTracker stops receiving heartbeat signals. Once the JobTracker notices a failed tasktracker, it removes the node from the list of tasktrackers to schedule tasks on. The tasks that are run on the failed node are re-run on other tasktrackers that are live and contain replicated copy of data for executing the job. The map tasks that are incomplete or completed successfully but belongs to a failed job are also re-run to make their output available for reduce tasks.

#### 2.4.2 Jobtracker Failure

Failure of JobTracker is considered as single point of failure as there is no mechanism for dealing with this failure. However JobTracker failure has very low chances of occurring. Once the JobTracker recovers from failure all the jobs that were running before failure must be re-run.

## 3. A New Resource Management Platform for Hadoop

To address the issues in Hadoop MapReduce, a new architecture is described that avoids scalability limitations, resource utilization problems and provide an enhanced computing platform for numerous programming models. In this approach the responsibilities of JobTracker are divided into two components: Resource Manager and Application Master. A component on each node, Node Manager holds the containers (cpu, bandwidth, memory) and launches the container as instructed by the Resource Manager.

### 3. 1 Resource Manager

The Resource Manager [2] controls the usage of resources across the cluster. The containers are allocated to the running applications based on resource requirements of the applications. The Resource Manager monitors the number of available nodes, resources and coordinates the allocation of resources to the applications. The scheduler that is in-built in Resource Manager performs its task of allocating resources to the applications. The decision of allocation is based on constraints such as queue capacity, fairness. The Resource Manager allocates resources to the Application Master and also keeps track of the applications on the Node Managers. Resource Managers adopts two kinds of scheduling policies according to the requirements, fair scheduling and capacity scheduling [4]. In fair scheduling the jobs are scheduled in such a way that each user gets a share of cluster capacity. The capacity scheduler maintains a cluster in the form of queues, and the jobs in the queue are schedule based on FIFO scheduling.

### 3. 2 Application Master

The Application Master [2] takes the role of TaskTracker of classic MapReduce. The Application Master has control on all the applications that runs in the cluster. This component is responsible for requesting resources from Resource Manager and coordinating the execution of all the tasks within the application. The Application Master also monitors the resource usage, resource allocation. The applications are run by Application Master using the containers that are controlled by Node Managers. Whenever a job is submitted by the client and the Resource Manager starts an Application Master. The Application Master keeps track of job's status and decides how to run the tasks. If the job is small with less number of mappers and reducers,it is run on same machine otherwise the Application Master requests more containers from Resource Manager. Once the container is assigned the Appliction Master starts the task by interacting with Node Manager.

### 3. 3 Node Manager

The Node Manager [2] is responsible for managing the nodes in a cluster. The Node Manager monitors the container resource usage by each node, health of the nodes. The Node Manager manages containers that represent resources used by applications in contrast to MapReduce map and resduce slots. The number of containers depends on the configuration parameters set. The Node Manager also ensures that the application does not use more than the allocated resources. yarn. nodemanager. resource. memory. mb, yarn. nodemanager. resource. cpu-vcores [8] properties are used to control the memory and cpu usage by each node.

### 3. 4 Improved Scalability

Hadoop architecture was constrained through the JobTracker, which was responsible for resource management and scheduling jobs across the cluster. YARN refines the responsibilities of the JobTracker to increase scalability. By breaking up the tasks, performed by JobTracker YARN eliminates many scaling issues of MapReduce. To enable greater scalability a hierarchical approach [6] on cluster framework was chosen. The YARN architecture replaces JobTracker with two new components: Resource Manager to manage the usage of resources across all applications, Application Masters which takes up the responsibility of managing the job execution. This approach removes the bottleneck and improves the ability to scale up the Hadoop clusters to a much larger configuration than it was previously possible.

### 3. 5 Enhanced Resource Utilization

The new architecture makes no difference between resources available for map tasks, resources available for reduce tasks. The default memory allocations for map and reduce tasks are avoided in the enhanced approach by using memory based scheduling. Problems of over and under utilization of resources is eliminated by replacing the allocation of slots to the nodes with containers, which are fine-grained. Each node

is configured with a set of memory, cpu[8] for executing their tasks. Nodes are configured with a number of virtual cores, which are resources meant to represent parallelism. In almost all cases, a node's virtual core capacity on the Node Manager should be set as the number of physical cores on the machine**.**

Applications may request a memory capability between the minimum allocation and a maximum allocation. Default memory allocations are scheduler-specific, and for the capacity scheduler the default minimum and maximum are 1024 MB, 10240 respectively. The tasks can obtain memory between these two values by setting *mapreduce. map. Memory. mb, reduce. reduce. memory. mb*[8] appropriately. This approach drastically improved resource utilization as the resource needs and capacity can be balanced easily. This approach also enables the sharing of resources between MapReduce and other frameworks, allowing more sensible and finer-grained resource configuration for better cluster utilization.

### 3. 6 Non-Mapreduce Workloads

Mapreduce is considered as one of the instance of YARN application. By decoupling resource management and scheduling tasks from JobTracker, YARN provides support for more varied processing approaches and a broader array of applications. Routines that repeat many times across a data set, such as machine learning algorithms or interactive real time Business Intelligence applications, processing large data streams where immediate computation is required, attempts to define a parallel processing model for graph processing that can work on Hadoop-style clusters are being implemented on YARN. YARN can be used for the creation of new frameworks and execution models that can leverage both the compute power of an Hadoop cluster and its rich data storage models to solve specific new classes of problems.

### 3. 7 High Availability

YARN is more available compared to classic MapReduce by eliminating the single points of failure.

#### 3. 7. 1 Application Master Failure

In the event of Application Master Failure [4], the Resource Manager fails to receive heartbeats. Once the Resource Manager detects the failure, a new instance of Application Master running on new container is started. The state of MapReduce tasks on the failed Application Master can be recovered avoiding re-running them. The client interacting with Application Master for application status, contacts Resource Manager to locate the new instance of Application Master in the case of failure.

#### 3. 7. 2 Node Manager Failure

In the case of Node Manager failure[4] failure, it stops sending heartbeats to the resource manager. The Resource Manager removes the failed Node Manager from the list of available nodes. The applications on the failed Node Manager can be recovered.

#### 3. 7. 3 Resource Manager Failure

Failure of Resource Manager [4] is the most serious issue since the jobs or containers can't be launched without it. This issue is overcome by checkpoint mechanism by saving the state of Resource Manager to persistent storage. On recovery from failure the Resource Manager's state contains the Node Managers and running applications and the Application Master checkpoints completed tasks, so the completed tasks need not be re-run.

### 3. 8 Job Submission Process

When a client submits an application, the Resource Manager launches Application Master to run the application. The Resource Manager maintaining the status of available resources initiates the scheduler to decide the allocation strategy. The Application Master once launched, registers with Resource Manager and requests the Resource Manager for containers required for application execution. Once the resources are allocated the Application Master interacts with Node Manager for launching the required containers. When the application execution completes the Application Master releases all the allocated resources, de-registers with Resource Manager.

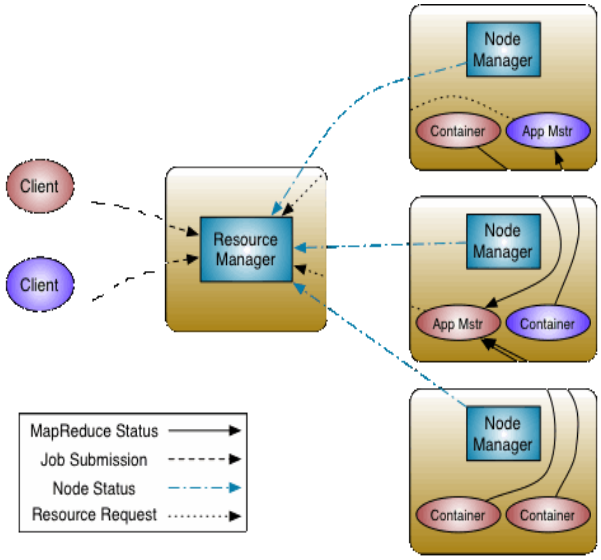


**Figure 2:** Running applications on YARN

## 4. Experimental Results

The Hadoop TeraSort program was used to run GraySort and MinuteSort benchmarks[7], using Hadoop HDFS to store the input and output. The input data was generated with gensort version 1. 5. For GraySort, the data was 102. 5TB in size, spread across 1025 files each with 100,000,000,000 bytes. For the Indy MinuteSort, the data was 1612. 22GB (1612223312700 Bytes) in size, spread across 1001 files each with 1610612700 bytes. For Daytona MinuteSort, the data was 1497. 86 GB (1497869841679 Bytes) in size, spread across 920 files each with 1610612700 bytes. Both

skewed and non-skewed data were generated as needed. The experiments were run on 2100 nodes each equipped with two 2. 3Ghz hexcore Xeon E5-2630, 64 GB memory, and 12x3TB disks each. This is shown in Table1.

The benchmarks on a 260 node cluster comparing MapReduce with YARN [2] is provided in Table2. Each node is running 2. 27GHz Intel(R) Xeon(R) CPU totaling to 16 cores, has 38GB physical memory, and 6x1TB 7200 RPM disks each, formatted with ext3 file-system. The network bandwidth per node is 1Gb/sec. Each node runs a DataNode and a Node- Manager with 24GB RAM allocated for containers. 6 maps and 3 reduces are run in 1. 2. 1, and 9 containers in 2. 1. 0.

**Table 1**: Daytona and GraySort benchmarks

| Benchmark | Data Type | Amount of Data Sorted | Time | Rate | Duplicate Keys |
|---|---|---|---|---|---|
| Daytona GraySort | non-skewed | 102.5 TB | 72 minutes 8.053 seconds | 1.42 TB per minute | 0 |
| Daytona GraySort | skewed | 102.5 TB | 117 minutes 48.261 seconds | 0.87 TB per minute | 31867643140 |
| Indy MinuteSort | non-skewed | 1612.22 GB | 58.027 seconds | | 0 |
| Daytona MinuteSort | non-skewed | 1497.86 GB | 59.223 seconds | | 0 |
| Daytona MinuteSort | skewed | 1497.86 GB | 1 minute 27.242 seconds | | 0 |

Each map occupies 1. 5GB JVM heap and 2GB total memory, while each reduce takes 3GB heap 4GB total. JobTracker/ResourceManager run on a dedicated machine so is the HDFS NameNode.

**Table 2:**. Hadoop benchmarks results

| Benchmark | Avg runtime (s) | | Throughput(GB/s) | |
|---|---|---|---|---|
| | 1.2.1 | 2.1.0 | 1.2.1 | 2.1.0 |
| RandomWriter | 222 | 228 | 7.03 | 6.84 |
| Sort | 475 | 398 | 3.28 | 3.92 |
| Shuffle | 951 | 648 | - | - |
| AM Scalability | 1020 | 353/303 | - | - |
| Terasort | 175.7 | 215.7 | 5.69 | 4.64 |
| Scan | 59 | 65 | - | - |
| Read DFSIO | 50.8 | 58.6 | - | - |
| Write DFSIO | 50.82 | 57.74 | - | - |

## 5. Conclusion

Hadoop continues to grow as one of the popular distributed data processing frameworks in big data market. However a number of issues are yet to be addressed and resolved to make hadoop suitable for large scale data workloads. YARN has evolved as a next generation compute platform providing greater scalability, availability, sharing on Hadoop cluster. YARN is the re-architecture of Hadoop that is responsible for managing and monitoring workloads, maintaining a multi-tenant environment, implementing security controls. YARN provides significant advantages on classic MapReduce, allowing the development of new distributed applications beyond MapReduce.

## 6. Future Work

YARN is still undergoing major improvements to bring multi-workload capabilities to Hadoop. As YARN provides flexibility for emerging data access, data management, security and integration tools, new projects such as Spark, Storm, Tez, Knox are being integrated into Hadoop.

## References

[1] Apache hadoop. http://hadoop. apache. org.
[2] V. K. Vavilapalli, A. C. Murthy, C. Douglas et al. , “Apache hadoop yarn: Yet another resource negotiator,” in Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013.
[3] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Proceedings of the 6th Symposium on Opearting Systems Design & Implementation, 2004.
[4] T. White, Hadoop: The definitive guide. ” O’Reilly Media, Inc.”, 2012.
[5] HADOOP YARN:http://hortonworks. com/hadoop/yarn [6] T. Graves. GraySort and MinuteSort at Yahoo on Hadoop 0. 23 http://sortbenchmark. org/Yahoo2013Sort. pdf, 2013.
[6] Karthik Kambatla, Wing Yew Poon, and Vikram Srivastava “How Hadoop YARN HA works “ http://blog. cloudera. com/blog/category/yarn “how-yarn-overcomes-mapreduce-limitations-in-hadoop-2-0” http://saphanatutorial. com/
[7] Dan Sullivan “Hadoop 2 vs. Hadoop 1: Understanding HDFS and YARN” www. tomsitpro. com/articles/hadoop-2-vs-1,2-718. html

[8] James Kobielus “YARN unwinds MapReduce's grip on Hadoop” http://www. infoworld. com/
[9] O'Reilly Strata “An Introduction to Hadoop 2. 0: Understanding the New Data Operating System” http://radar. oreilly. com
[10] http://ibm. com/developerworks/library/bd- hadoop yarn
[11] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. ” Mesos: a platform for fine-grained resource sharing in the data center”. In Proceedings of the 8th USENIX conference on Networked systems design and implementation, NSDI’11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.
[12] M. Schwarzkopf, A. Konwinski, M. Abd-El- Malek, and J. Wilkes. “Omega: flexible, scalable schedulers for large compute clusters”. In Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys ’13, pages 351–364, New York, NY, USA, 2013. ACM.
[13] R. Chaiken, B. Jenkins, P. -A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. “Scope: easy and efficient parallel processing of massive data sets”. Proc. VLDB Endow. , 1(2):1265–1276, Aug. 2008.

## Author Profile

**G. Prabhakar Raju** received the M. C. A degree from Osmania University and M. Tech degree in Computer Science and Engineering from JNTU University. He is an associate professor in the Department of Computer Science and Engineering, Anurag Group of Institutions. His research interests include data mining, data warehousing, bigdata, hadoop, natural language processing and artificial intelligence.

**Sandeep Raj R** received the B. Tech degree in Information Technology from JNTU Hyderabad. in 2012. He is pursuing M. Tech in Computer Science from JNTU Hyderabad. His research interests include bigdata, hadoop and cloud computing.