Comparison of Resource Management Frameworks for Processing Big Data

Sanam Ahmad, Muhammad Murtaza Yousaf, Syed Waqar Jaffry, Shahzad Sarwar, and Laeeq Aslam

***Abstract* – Big data analytic solutions are usually deployed on computer-cluster or cloud computing environment. Efficient resource management is a challenging task in a cluster or cloud environment. Big data processing frameworks hide the details of resource management from the end user. In this paper, we have described the traditional Hadoop and state-of-the-art resource management frameworks used for big data processing. For the evaluation of resource management framework we have identified a feature vector comprising resource assignment, scheduling, security, and physical system-level resources. Using the feature vector, resource management frameworks have been compared in order to identify strengths and weaknesses of each. Identified weaknesses indicate the future research directions for further improvement of respective framework. It has been observed that YARN framework qualifies for the most features in the feature vector.**

***Index Terms* – Resource management framework, big data processing frameworks, Hadoop, MapReduce, Corona, YARN, Mesos, MROrchestrator**

I. INTRODUCTION

In the digital world, amount of data is increasing tremendously. Data has hidden information that can add value to a business. To process data that is being produced in high velocity, variety, and volume is a challenging task. Efficient data processing frameworks are being designed that facilitate the data processing. The power of computing nodes, also termed as cluster of computing nodes, is harnessed to process big data. Big data processing frameworks hide the details of data distribution, parallelization, fault tolerance, and scalability from application developers. To process data, cluster resources such as processing power, storage, memory, and bandwidth are used. Data processing frameworks like Hadoop implementation of MapReduce also manage the cluster resources. In this study we have explored the resource management frameworks that facilitate the big data processing frameworks.

Generally applications have versatile processing needs that cannot be met by a single data processing framework. Due to the versatility, novel frameworks for big data processing are being designed. The resource utilization of a cluster increases when multiple data processing frameworks are deployed on it. Resource management framework is required to let the cluster resources be shared among concurrent data processing frameworks. Resource management framework uses either the following two approaches:

1. Push-based resource assignment: Resource management framework assigns resources to a big data processing framework.
2. Pull-based resource assignment: Big data processing framework requests and pulls resources from a resource management framework.

We have discussed Hadoop implementation of MapReduce framework and scheduling limitations in Section II. Further, we have described state-of-the-art resource management frameworks in Section III. A feature vector has been identified for the comparison of resource management frameworks in Section IV. Section V concludes the discussion.

II. HADOOP MAPREDUCE FRAMEWORK

Hadoop is an open source MapReduce implementation provided by the Apache Software Foundation. It uses Hadoop distributed file system (HDFS) as the underlying file system. In Hadoop MapReduce implementation master process is called job tracker. The slave process is called as task tracker. Nodes on which task tracker are running are called slave nodes. Each slave node is assigned a fixed number of map/reduce slots based on configuration or the number of cores in the node.

The MapReduce framework is based on functional programming. There are two primary functions of this framework: map- and reduce-function. The MapReduce framework [1] processes data as key-value pairs. The map-function takes key-value pair as input and generates intermediate key-value pairs. The reduce-function uses the intermediate key-value pairs generated by the map task and reduces all values associated with same intermediate key to a single value. The MapReduce framework hides details of data distribution and communication among tasks from application developers. Also, it exploits parallelism and provides fault-tolerance, as well.

The task tracker can execute map- or reduce-functions according to the instructions of master process. The master process periodically pings all the worker nodes to detect failure. If worker does not respond in threshold time, it is marked as failed and no more tasks would be scheduled on it. Master reschedules all map tasks previously assigned to failed nodes. Reduce worker writes to a temporary file and on completion renames its temporary file to final output name. The global file system handles the rename operation by multiple processes. File system divides each file into 64 MB chunks and stores multiple copies of each chunk on different machines. The master process schedules map tasks closer to the location of input file in order to reduce latency

Sanam Ahmad, Muhammad Murtaza Yousaf, S. Waqar Jaffry, Shahzad Sarwar and Laeeq Aslam are with Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan.
Email: sanam.ahmad@pucit.edu.pk, murtaza@pucit.edu.pk, swjaffry@pucit.edu.pk, s.sarwar@pucit.edu.pk, laeeq.aslam@pucit.edu.pk.
Manuscript received Oct 02, 2016; revised on Nov 18, 2016; accepted on Dec 15, 2016.

 ISSN – 1999-4974

and to conserve network bandwidth. First, it tries to schedule a map task on machine containing a replica of the corresponding data. If the machine containing replica is not idle, then it looks for another idle machine in the rack, then in the same network switch and so on for scheduling map task. MapReduce framework schedules multiple copies of a task if there are free slots/resources available. If any copy of the task is completed then task is marked as completed.

A. Hadoop resource management and scheduling limitations

The Hadoop cluster has one job tracker and many task trackers. Job tracker has many responsibilities like resource management, job scheduling, job status management, and fault-tolerance. The job tracker is overloaded with all these responsibilities that results in performance deterioration under peak load scenarios [3].

The Hadoop task scheduling follows a pull-based model in which the task tracker sends periodic *heartbeat* messages to job tracker. The messages contain information of available resources. Afterwards, the job tracker assigns tasks to a task tracker. The intervals of *heartbeat* messages are configurable. Setting small *heartbeat* intervals overload job tracker while long *heartbeat* periods causes scheduling delay. The Hadoop MapReduce cluster nodes have static number of map- and reduce-slots. If workload does not fit into this static slot configuration then resources would be wasted. For example, if available slots are map-slots then reduce phase of job would be delayed until the reduce-slots are available.

B. Competitors of Hadoop

There are many data processing frameworks that are being used in industry and academia for big data processing. Strom [12] is an open source technology to process real-time streams of data. Storm enables real-time processing mode for big data processing as compared to Apache Hadoop that follows batch mode of processing. The Apache Tez makes MapReduce paradigm more powerful framework for executing a complex directed acyclic graph (DAG) of tasks [13]. The Tez meets demands for fast response times and extreme throughput for petabyte data.

S4 is an open source, distributed, scalable, and fault-tolerant framework for processing continuous streams of data [5]. Esper is an event stream processing (ESP) and event correlation engine written in java [14]. Esper performs actions against event conditions in real-time streams.

There are different data processing frameworks that are being designed to tackle the challenges posed by big data. Resource management frameworks are also being designed for data processing frameworks. Many resource management /scheduling algorithm or technologies are solely targeted towards Hadoop cluster resource management. Polo et al. proposed a scheduling algorithm for Hadoop jobs that can perform resource adjustments based on estimated completion times of the jobs [15]. A kernel level virtualization technique has been proposed to reduce resource contention among MapReduce jobs [16]. Hadoop is a major but not the only data processing framework, there is need to design efficient resource management frameworks that can manage the resources for variety of frameworks simultaneously. In this study we have summarized resource management frameworks that perform resource management for multiple data processing frameworks simultaneously and increase the resource utilization of the clusters.

III. RESOURCE MANAGEMENT FRAMEWORKS

In this section, resource management frameworks, including Corona, YARN, Mesos, and MROrchestrator are explored. In the following subsections details of each framework are presented.

A. Corona

The Corona cluster manager monitors the cluster nodes and schedules data processing jobs as per available resources. The Corona resource manager does not monitor the job progress. The job tracker requests resources from cluster manager. The Corona cluster manager provides resource grants to job tracker while Job tracker assigns tasks to task trackers. The Corona cluster manager provides resources dynamically in pull-based style to job trackers. The approach of Corona's job tracker is simpler as compared to that of Hadoop. The task trackers perform assigned tasks. The task trackers do not need to send *heartbeat* messages as they just have to perform assigned tasks and send their task completion status to job tracker. The Corona has improved implementation of fair share scheduling. The system wide resource and job provision of Corona provides better implementation of fair scheduling.

The job tracker requests resources from resource manager. On receiving resource grants it creates tasks and pushes these tasks to task trackers. In traditional Hadoop, task trackers send *heartbeat* messages to job tracker and pull tasks from job trackers. Resources are pulled and tasks are pushed in the Hadoop map-reduced system managed by Corona. The Corona resource manager is well tested and proved successful in Facebook even in peak loads. Corona design should be optimized to include the support of multiple frameworks. So performance of other frameworks like Storm [8] and S4 [5] could be boosted like Hadoop with Corona resource manager.

1) Lack of Authentication mechanism

In cloud environment Corona resource manager provides services to different data processing job trackers. Corona currently does not perform any authentication while assigning resources to job trackers as discussed in [3]. Corona should authenticate the resource assignment requests by job trackers as a fake job tracker could request resources and cause resource wastage and performance degradation of jobs.

B. YARN

The Apache Hadoop YARN provides two level of resource management [6]: cluster and node level resource managers named as resource-manager and node-manager, respectively. The node-manager manages the resources of a

 ISSN – 1999-4974

single node. The resource-manager uses the services of node-managers to manage all resources in a cluster. Efficient resource management is the primary objective of a resource-manager. It schedules memory, CPU, disk, and network bandwidth to meet the requirements of an application.

Another important component of this framework is application master that is framework/application specific instance to manage the execution of a job. The application master is responsible for job execution in fault-tolerant way. In general, application specific monitoring is the responsibility of application master.

YARN is designed to provide resource management to variety of processing frameworks including MapReduce, MPI, SPARK, and S4. Further, it has pluggable scheduling feature which accommodates MapReduce schedulers: the capacity scheduler and the fair scheduler. YARN employs a pull-based resource assignment in which an application master requests resources from resource manager. Then resource manager provides these resources to the application master. The application master is authenticated prior to resource assignment. The requested resources are assigned in the form of a container. YARN assigns resources to application master which provides container assignment to the node manager of the host on which containers are located. The node-manager further verifies this container assignment, making it a distinctive security feature.

The application master specifies the request parameters to optimize data locality and performance of the jobs. The description of four parameters of request is as follows:

1. Resource requirement: The amount of CPU and memory required.
2. Number of containers: The required number of containers.
3. Resource-name: The name of required resource host name, rack number, or "*" if host name and rack number is not known.
4. Priority: Relative priority of resource allocations in comparison of requests by the same application master.

C. Mesos

Mesos [2] is a resource management framework that performs resource management for different cluster computing frameworks like Hadoop and message passing interface (MPI). It has a simple architecture that enables easy extension and improvement. After deciding what resources can be offered to the processing frameworks, it offers or pushes those resources to the frameworks. The processing framework may accept or decline the offer. If the resource offer does not meet constraint of data locality, a processing framework would decline the offer. Thus, scheduling of the tasks would be delayed, making it a downside of push-based approach.

It is scalable as adding nodes in cluster does not degrade its performance. Fault tolerance is achieved by using zookeeper [9]. The master resource manager processes maintains information of failed nodes and available resources which is critical for resource allocation policy. A Mesos daemon, running on the slave nodes keeps informing about the free resources to the master resource manager. The configuration of master resource manager is maintained at the zookeeper [9] so that in case of a master failure, a fresh master instance could be instantiated using the soft-states.

Scheduling requirements of the processing frameworks are met by delegating scheduling control to them. The processing framework schedules its tasks on the accepted resources. Its pluggable resource allocation module enables an organization to implement its resource allocation policy.

The allocation module guarantees a specific amount of resources to each framework. It associates a time limit for the resource usage. In order to resolve contention for resources, a task would be killed if it keeps the resource longer than its time limit.

Mesos allows plug-gable isolation mechanism that isolates tasks of different processing framework running on the same slave node. For this purpose, it requires support mechanism from the host operating system. One such mechanism is Linux container.

Mesos allows a processing framework to describe its resource requirements. Correspondingly, it makes a resource offer to the processing framework. If the framework does not respond to the offer in a specified time, the resource-offer is discarded.

D. MROrchestrator

This resource management framework improves the performance of map-reduce jobs by assigning them the required resources [4]. The resource contention is resolved by precise estimation and assignment of resources. MROrchestrator framework is comprised of the following components:

- Local resource-manager (LRM): This component runs on map and reduce task-trackers. Local resource manager has further following components:
  - Resource profiler: For every task in task-tracker, it monitors resource utilization and embeds this information in *heartbeat* message for the consumption of the job-tracker.
  - Estimator: It creates statistical estimates based on the utilization of allocated resources.
- Global resource-manager (GRM): It runs on job-tracker. LRM updates contention detector of GRM with resource imbalance; consequently, performance balancer in GRM suggests balanced resource allocation to LRM.

LRM uses Hadoop profiling [7] to monitor resource utilization and allocation for the task and embed this information in the *heartbeat* message of the task. Contention detector module of GRM running on job-tracker receives *heartbeat* messages to get resource profiling information. The GRM uses this profiling information to identify resource deficient and resource abundant tasks. The estimator uses predictive models to calculate resource finish time as function of resource usage. The difference between estimated and actual resource usage is the resource imbalance. This resource imbalance is provided to GRM so that it may adjust it.

ISSN – 1999-4974

The task progress score (PS) and actual finish time (AFT) is used to calculate the estimated finish time (EFT) based on a scaling factor α according to the equation 1.

$$EFT = \alpha \times \left(\frac{1-PS}{PS}\right) \times AFT \quad (1)$$

The LRM performs resource estimation after specific interval using the following two statistical approaches: uniform and regression.

GRM receives resource estimations and resource imbalances from all tasks and use these resource estimations to dynamically increase allocated resources of the tasks after evaluation in global view of all tasks. It has information about task placements on task trackers and resource deficient and resource abundant tasks.

Let us consider a scenario as shown in Fig. 1 where we have three nodes N_1, N_2, and N_3. Further, consider three jobs A, B and C, each having two map-tasks. The set of map-task is $\{M_{ij} \mid i \in (A,B,C), j \in (1,2)\}$. For example, M_{A1} represents first map task of job A.

Assigned map-tasks to each node are shown in Fig. 1. As per the scenario, M_{A1} and M_{C2} are CPU deficient; M_{B2} is RAM deficient, while M_{A2}, M_{B1}, and M_{C1} are resource abundant. GRM detects that between M_{B2} and M_{C1} resource contention is for memory (RAM) and resource contention between M_{A1} and M_{B1} is for CPU so it increases CPU allocation of M_{A1} and increases memory allocation of M_{B2}. Having information about resource balance of each task GRM can intelligently take decision to improve overall performance of all jobs.

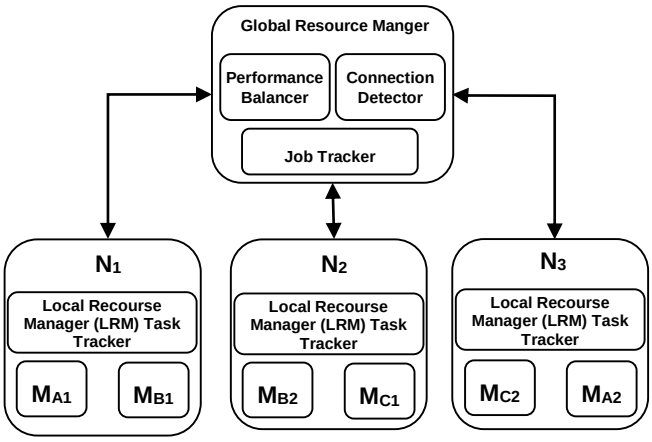

Fig. 1 MROrchestrator: Scenario of resource distribution

IV. COMPARISON OF RESOURCE MANAGEMENT FRAMEWORKS

In this section we present a feature vector, comprising resource assignment, scheduling, security, and physical system-level resources. Using the feature vector, a comparison of resource management frameworks is presented in Table 1. The description of each parameter of the feature vector is given in the following subsections.

A. *Support for multiple processing frameworks*

The support of multiple data processing frameworks is crucial because it increases the resource utilization. YARN and Mesos provide sharing of cluster resources among multiple data processing frameworks.

B. *Pluggable-scheduling*

Scheduling algorithm schedules the resource. There is variety of scheduling algorithms like FIFO, priority based, and fair share algorithms. Each algorithm has its own advantages and disadvantages. A resource management framework with pluggable scheduling algorithm can tailor scheduling approach according to requirement. For example, if fair sharing of resources is required fair share algorithm can be plugged in. Whenever prioritization of assignments is required, priority queue algorithm can be used.

Table I. Comparison of Resource Management frameworks

		Resource Management Frameworks			
Sr. No.	Feature Vector	Corona	YARN	Mesos	MROrchestrator
1.	Multiple frameworks Resource-Management	No	Yes	Yes	No
2.	Pluggable-Scheduling	No	Yes	Yes	No
3.	Production Status	Yes	Yes	Yes	No
4.	Push-based Resource Assignment	No	No	Yes	No
5.	CPU Management	Yes	Yes	Yes	Yes
6.	Memory management	Yes	Yes	Yes	Yes
7.	Network-Bandwidth Management	Yes	No	No	No
8.	Disk Management	Yes	Yes	No	No
9.	Security Measures	No	Yes	No	No

C. *Production status*

The production status of the resource management frameworks is the metric to check which frameworks are under experiment and which are being used in production environment. As discussed by [3, 11], Corona is used by Facebook in production environment, YARN is used by Yahoo in production environment. Mesos is used in production environment of Twitter, eBay, and Apple [10].

D. *Push-based resource assignment*

A recent approach is to push resources to frameworks prior to their request. It is interesting to know which resource management frameworks are using this approach. By pushing resources, it avoids issues due to variation in the intervals of *heartbeat* message.

E. CPU, memory, disk, and network bandwidth management

For processing, the core cluster resources are CPU, memory, disk, and network bandwidth which are shared among data processing frameworks.

ISSN – 1999-4974

F. *Security measure*

While assigning resources, security measures, ensures the authentication of data processing framework, in order to avoid resource wastage.

V. CONCLUSION

Big data processing frameworks hide the details of resource management from the developers. We have compared multiple resource management frameworks using the feature vector. Mesos is only framework that follows push-based resource assignment approach. YARN framework qualifies for the most features in the feature vector. Corona is a cluster manager which is good for high workloads but lacks security measures. MROrchestrator is relatively new cluster management framework that provides CPU and memory management.

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Communications of the ACM, 51 (1): 107-113, 2008.

[2] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica, "Mesos: Flexible Resource Sharing for the Cloud", proceedings of USENIX , August 2011.

[3] Notes by Facebook engineering, "Under the Hood: Scheduling MapReduce jobs more efficiently with Corona", November 9, 2012 online: http://www.facebook.com/notes/facebook-engineering/under-the-hood-scheduling-mapreduce-jobs-more-efficiently-with-corona/10151142560538920

[4] Bikash Sharma, Ramya Prabhakar, Seung-Hwan Lim, Mahmut T. Kandemir, and Chita R. Das, "MROrchestrator: A Fine-Grained Resource Orchestration Framework for MapReduce Clusters", Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, pp.1-8, June 24-29, 2012.

[5] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari, "S4: Distributed Stream Computing Platform", Proceedings of the 2010 IEEE International Conference on Data Mining Workshops, pp.170-177, December 13-13, 2010.

[6] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler, "Apache Hadoop YARN: yet another resource negotiator", Proceedings of the 4th annual Symposium on Cloud Computing, October 01-03, 2013, Santa Clara, California.
Apache.Hadoopprofiler: https://issues.apache.org /jira/browse /MAPREDUCE-220.https://storm.apache.org/

[7] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed, "ZooKeeper: wait-free coordination for internet-scale systems", USENIXATC'10: Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIX Association, 2010.

[8] http://container-solutions.com/the-rise-of-apache-mesos/.

[9] http://developer.yahoo.com/blogs/ydn/hadoop-yahoo-more-ever-54421.html

[10] http://storm-project.org/

[11] http://hortonworks.com/Hadoop/tez/

[12] http://esper.codehaus.org/

[13] Jordà Polo, David Carrera, Yolanda Becerra, Malgorzata Steinder, and Ian Whalley, "Performance-driven task co-scheduling for MapReduce environments", In 12th IEEE/IFIP Network Operations and Management Symposium. ACM, 2010.

[14] An Qin, Dandan Tu, Chengchun Shu, and Chang Gao, "XConveryer: Guarantee Hadoop throughput via lightweight OS-level virtualization, " In Proceedings of 8th International Conference on Grid and Cooperative Computing, pp.299-304, 2009.

[15] Hadoop.CapacityScheduler:http://hadoop.apache.org/common /docs /r0.19.2/capacityscheduler.html

[16] Hadoop:.FairScheduler:.http://hadoop.apache.org/common/docs/r0.20.2/fairscheduler.html

[17] Thomas Sandholm, Kevin Lai, "Dynamic proportional share scheduling in Hadoop", Proceedings of the 15th international conference on Job scheduling strategies for parallel processing, p.110-131, April 23, 2010, Atlanta, GA.

[18] .Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker and Ion Stoica,."Job Scheduling for Multi-User MapReduce Clusters", EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2009-55, April 30, 2009.

 ISSN – 1999-4974