PRISM: Fine-Grained Phase and Resource Information-aware Scheduler for Map-Reduce

Swati R. Mahendrakar
PG Student
M B E Society's College of Engineering,
Ambajogai
Maharashtra, India

B. M. Patil
Professor
M B E Society's College of Engineering,
Ambajogai
Maharastra, India

ABSTRACT
In recent years, Map Reduce has become a popular model with regard to data-intensive computation. Map Reduce can significantly reduce the execution time of data-intensive jobs. In order to achieve this objective, Map Reduce breaks down each job into small map and reduce tasks and executes them in parallel across a large number of machines. However, existing solutions mainly focus on scheduling at the task-level, which offer sub-optimal job performance, because tasks may have resource requirements which may vary during their lifetime. This makes it difficult for existing system's task-level schedulers to effectively utilize available resources in order to reduce job execution time.

To avoid this limitation, PRISM is introduced. PRISM stands for Phase and Resource Information-aware Scheduler for Map-Reduce. PRISM consists of various clusters that perform resource-aware scheduling at the level of phases. PRISM can be defined as a fine-grained resource-aware Map Reduce scheduler that divides tasks into phases. Here, each phase has a constant resource usage profile, so that not a single phase suffers from starvation. PRISM also offers high resource utilization and provides 1:3x improvements in job running time as compared to the current Hadoop schedulers.

Keywords
Map Reduce, scheduling, resource allocation.

1. INTRODUCTION
Now-a-days, businesses are entirely dependent on large-scale data analytics so that, they can make critical day-to-day business decisions. This turns towards the development of Map-Reduce, i.e., a parallel programming model which has become equivalent with large-scale and data-intensive computations. Map-Reduce comprises of a job, which is a collection of Map and Reduce tasks. These tasks can be scheduled synchronously on multiple machines, which results in substantial reduction in job running time.

An essential component of a Map-Reduce system is its job scheduler. The main role of job scheduler is to create a schedule of Map and Reduce tasks that spans one or more jobs, minimizes job completion time and maximizes resource utilization. In many situations, heavy resource contention and long job completion time occurs due to a schedule with too many simultaneously running tasks on a single machine. On the contrary, starvation occurs due to the poor resource utilization and also due to a schedule with too few concurrently running tasks on a single machine.

The problem of job scheduling becomes considerably easier to solve, if there is an assumption that all map tasks (and likewise, all reduce tasks) has consistent resource requirements for example, CPU, memory, disk and network-bandwidth. However, this assumption is used to simplify the scheduling problem by the current Map-Reduce systems, such as Hadoop Map-Reduce Version 1.x. This system uses a simple slot-based resource allocation scheme, in which the physical resources on each machine are seized by the number of indistinguishable slots that can be allocated to tasks.

This paper offers PRISM, i.e., a Fine- grained Phase and Resource Information-aware Scheduler for Map-Reduce clusters. PRISM accomplishes resource-aware scheduling at the level of phases. Precisely, this paper shows that for utmost Map-Reduce applications, the task resource consumption during run-time can vary considerably from phase to phase. Therefore, it is possible for the scheduler to succeed higher degrees of parallelism although avoiding resource contention, only by taking care of the resource demand at the phase level. Hence, by the end, this paper has developed a phase-level scheduling algorithm with the aim of attaining high job performance along with proper resource utilization.

2. LITERATURE SURVEY
This section provides an overview of various studies and surveys, which is related to PRISM.

2.1 Job-Scheduling and Phases
A number of recent studies have conveyed that, often the production workloads have miscellaneous utilization profiles and performance requirements [8]. Deteriorating to consider these job usage characteristics can hypothetically lead to ineffective job schedules with low resource utilization and extended job execution time too. Inspired by the above observation, numerous recent proposals, such as resource-aware adaptive scheduling (RAS) [15] and Hadoop Map-Reduce Version 2 (also known as Hadoop NextGen and Hadoop Yarn) [7], have announced resource-aware job schedulers for the Map-Reduce framework. On the other hand, these schedulers insist on a fixed size for each task in terms of essential resources (e.g. CPU and memory). Thus, the run-time resource consumption of each task is constant over its life time.

A phase can be defined as a sub-procedure in the task that has a distinct determination and can be considered by the identical resource consumption over its duration. There are two types of Job-Scheduling, which are Task-level Scheduling and Phase-level Scheduling.

2.1.1 Task-Level Scheduling
In Task-level Scheduling, it is difficult for schedulers to effectively utilize the available resources to reduce job execution time, because tasks can have highly varying resource requirements during their lifetime. Subsequent phase of the task may not be scheduled simultaneously. Task level scheduling suffers from insufficient scheduler decision

making problem. There are inadequate resources problems too. Therefore, delay problem might occur. Overall, performance and efficiency of map reduce frameworks have become critical.

2.1.2 Phase-Level Scheduling

In Phase-level Scheduling, PRISM, i.e., a fine-grained resource-aware map-Reduce scheduler, divides tasks into phases, where each phase has a persistent resource usage profile and implements scheduling at the phase level. Here, by considering the resource demand at the level of phases, it is possible for the scheduler to succeed higher degrees of parallelism to avoid resource contention. Because of this parallel implementation there is enhancement in high Resource Utilization. However, this improves the job Running Time. Therefore, achieves high job Performance.

2.2 Map-Reduce Job Phases

Existing Hadoop job schedulers implement task-level scheduling, where tasks are considered as the supreme granularity for scheduling. However, if the execution of each task is examined, then it can be found that a task consists of multiple phases. In particular, a Map-Reduce job consists of two types of tasks, namely map and reduce tasks. A Map task takes as input a key-value block stored in the distributed file system. Subsequently, a Reduce task is responsible for collecting and applying a user-specified reduce function on the collected key-value to produce the final output.

In the map phase, when a mapper draws an input data block from the Hadoop Distributed File System [4] and applies the user-defined map function on each record, then those records are collected into a buffer. When the buffer becomes full, then the content of the buffer will be written to the local Input-Output disk. Finally, the mapper performs a merge phase to group the output records and store the records in multiple files, so that each file can be fetched a consistent reducer. In the same way, the implementation of a reduce task can be distributed into three phases: shuffle, sort, and reduce. In the shuffle phase, the reducer raises the output file from the local storage of each map task. Then, it places that file in a storage buffer that can be either memory or disk depending on the size of the content. When the buffer is fully occupied, then the content of the buffer will be written to the local Input-Output disk. At the same time, the reducer also inaugurates one or more threads to implement local merge sort in order to reduce the running time of the succeeding sort phase. As soon as, all the map output records have been collected, then the sort phase will perform an ultimate sorting procedure to confirm all collected records are in a specific order. Finally, in the reduce phase, the records are processed in the sorted order, and the output is written to the HDFS. Different phases may have different characteristics in associated to resource consumption. For instance, the shuffle phase often consumes substantial network I/O resources as it needs collecting outputs from all accomplished map tasks. In disparity, the map and reduce phases essentially process the records on the local machines. Therefore, they usually demand greater CPU resources than network bandwidth.

3. PRISM

3.1 Prism Architecture

As it is cleared from the definition that, PRISM is a resource information-aware Map Reduce scheduler that distributes tasks into phases in a fine-grained manner, where each phase has a persistent resource usage profile and implements scheduling at the level of phases. During the execution time of a task, resource usage analysis may lead to ineffective scheduling decisions. Because of this, at run-time, if the resource allotted to a task is higher than the existing resource usage, then the idle resources are wasted. On the other hand, if the resources allotted to the task is much less than the actual resource demand, then the resource can suffer from a situation called, bottleneck, which may slow down task execution.

Therefore, a fine-grained, phase-level scheduling mechanism has been introduced. This allocates the resources according to the demand of the phase that each task is currently executing. Due to this fine-grained resource allocation, not a single task suffers from either bottleneck or starvation problem.

An overview of the PRISM architecture is shown in Fig. 1. PRISM comprises of four main modules: resource manager, local node managers, a job progress monitor and a phase-based scheduler. Initially, Resource Manager (also known as a job tracker), is responsible for scheduling tasks on each local node. Then, Local Node Manager, (also known as a task tracker) that coordinate phase transitions with the scheduler. Next is Job Progress Monitor, which is responsible to capture phase-level progress information. Finally, Phase-Based Scheduler, i.e., a fine-grained, phase-level scheduling mechanism that allocates resources according to the demand of executing phase (neither overflow nor underflow).

3.2 Phase-Level Scheduling Mechanism

In this mechanism, there are some steps which are followed during the execution of PRISM. These steps are:

(Step 1): Each local node manager sends a heartbeat message to the phase-based scheduler periodically. As soon as a task requests to be scheduled, then the scheduler immediately responses to the heartbeat message with a task scheduling request.

(Step 2): Then, the local node manager initiates the task.

(Step 3): As and when a task completes implementing a particular phase (shuffle phase), then the task requests the local node manager for permission to start the next phase (e.g. reduce phase).

(Step 4): The local node manager then forwards this permission request to the phase-based scheduler.

(Step 5): Finally, once the task is permitted to execute the next phase (reduce phase), the local node manager grants permission to process that task and once the task is completed; the task status is received by the local node manager and then dispatched to the phase-based scheduler.

PRISM requires constant phase-level resource information for each job to perform phase-level scheduling. In this way, the entire task is implemented. Each phase travels through all the above steps and finally get completed successfully.

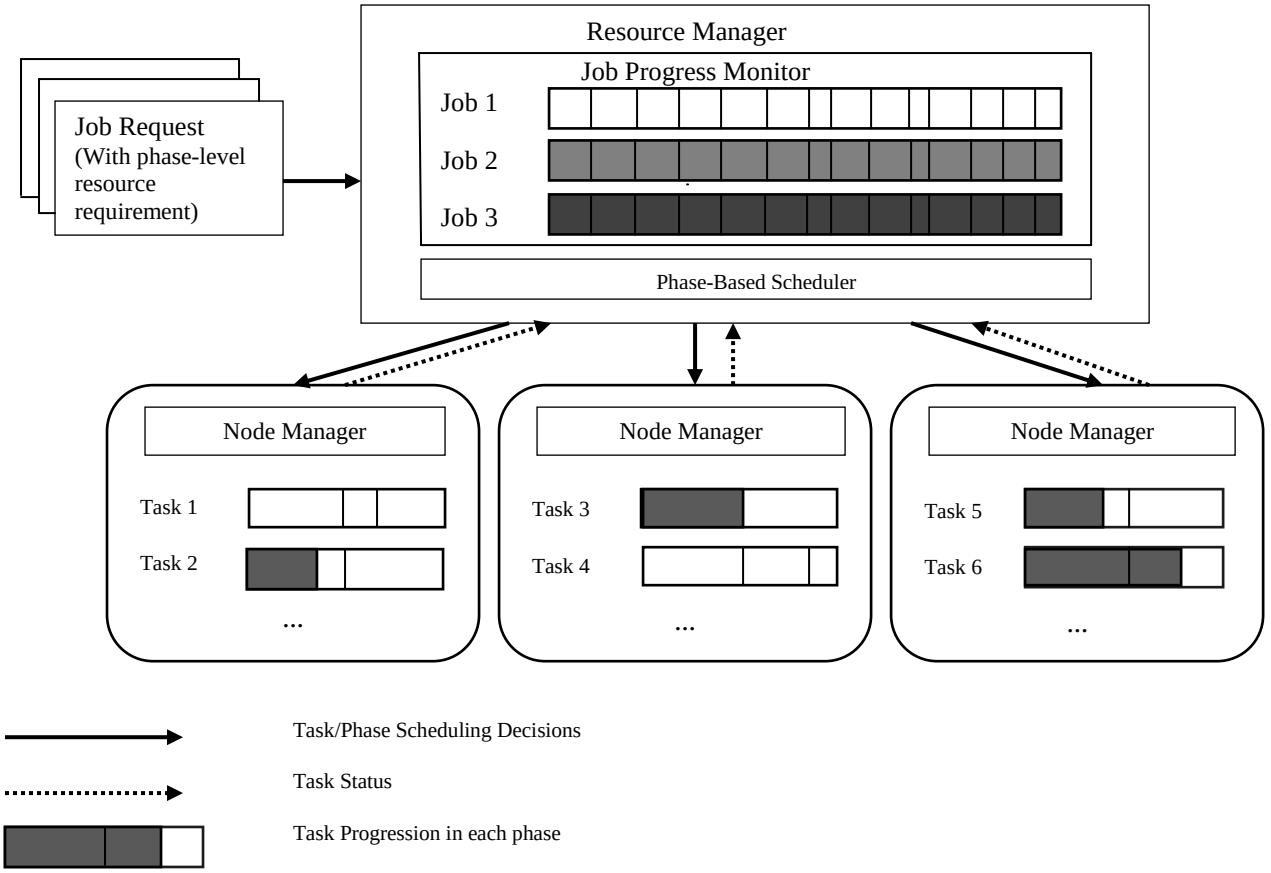

Fig. 1: System architecture [16]

4. ALGORITHM AND ITS DESCRIPTION

4.1 Phase-Level Scheduling Algorithm

1: Upon receiving a status message from machine n:
2. Obtain the resource utilization of machine n
3. PhaseSelected **PS** ← {∅}
4. CandidatePhases **CP** ← {∅}
5. **repeat**
6. **for** each job j ∈ jobs that has tasks on **n do**
7. **for** each schedulable phase i ∈ j **do**
8. CP ← CP ∪ {i}
9. **end for**
10. **end for**
11. For each job **j** ∈ top k jobs with highest deficit n **do**
12. **if** exist schedulable data local task **then**
13. CP ← CP ∪ {first phase of the local task i}
14. **else**
15. CP ← CP ∪ {first phase of the non-local task i}
16. **end if**
17. **end for**
18. **if** CP ≠ ∅ **then**
19. **for** i ∈ CP **do**
20. **if** *i* is not schedulable on *n* given current utilization **then**
21. CP ← CP ∪ {i}
22. Continue;
23. **end if**
24. Compute the utility **U** (i, n)
25. **if** U(i, n) <= 0 **then**
26. CP ← CP ∪ {i}
27. **end if**
28. **end for**
29. **if** CP ≠ ∅ **then**
30. **i** ← task with highest U (i ,n) in the CP
31. PS ← PS U {i}
32. CP ← CP ∪ {i}
33. Update the resource utilization of machine *n*
34. **end if**
35. **end if**
36. Until CP = = ∅
37. Return ***PS***

4.2 Algorithm Description

This algorithm describes the scheduling algorithm used by the phase-based scheduler. In this algorithm, two important concepts are used, which are Efficiency and Fairness [8], [14]. However, a Map-Reduce scheduler is responsible to assign each task to an appropriate machine along with the consideration of both Efficiency and Fairness.

Efficiency is achieved only by maintaining high utilization of resources in a cluster by the job schedulers. Another effective measure for efficiency is job running time because, during an execution of a task, minimum job running time indicates maximum utilization of resources in an efficient manner.

Secondly, Fairness provides an assurance that, all the resources are fairly distributed among each and every job. This aspect ensures that, there will be neither a bottleneck situation (i.e. overflow of resources) nor a starvation situation (i.e. underflow of resources).

However, achieving both the aspects, i.e. Efficiency and Fairness concurrently seems to be very challenging with respect to the multi-resource scheduling.

Initially in the above algorithm, the local node manager sends the status message to the Phase-Level Scheduling Algorithm in order to allocate necessary resources. Then, Line 2 states that, upon receiving the status message from a local node manager running on machine n, the algorithm computes the utilization u of the machine using job's phase-level resource requirement. Next, Lines 4-10 consists of a set of candidate phases (i.e. the schedulable phases) and selects these phases in an iterative manner.

Then, in Lines 11-23, for each job j, if phase has highest deficit n, then first phase of local task i is executed; otherwise first phase of the non-local task i is executed. These steps are iterated for each job.

Next, in Lines 24-28, for each schedulable phase i of each job j in each iteration, the algorithm computes the utility function U (i, n) according to the following equation:

$$U(i, n) = U_{fairness}(i, n) + a \cdot U_{perf}(i, n)$$

where,
$U_{fairness}$ and U_{perf} signifies the utilities for improving the fairness and job performance, respectively, and 'a' is an adjustable weight factor.

The fairness of each phase is calculated as

$$U_{fairness}(i, n) = U_{before\ fairness}(i, n) - U_{after\ fairness}(i, n)$$

where,
$U_{before\ fairness}$ and $U_{after\ fairness}$ signify the fairness measures of the job before and after scheduling phase i on machine n.

Then, in Lines 29-32, the phase with the highest utility for scheduling is selected and Line 33 updates the resource utilization of the machine n.

Subsequently, in Lines 34-37, the algorithm repeats the above steps by re-computing the utility of all the phases in the candidate set, and selects the succeeding best phase to schedule. Finally, the algorithm concludes when the candidate set becomes empty, which means that, there is no suitable phase to be scheduled.

5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

5.1 Implementation

The PRISM architecture is implemented as both, i.e. existing and proposed systems. These systems are simulated in Net-Beans IDE version 8.1 simulator. All the implementation and experiments are performed on a machine running Microsoft Windows XP operating system. The algorithm for executing the project is implemented in Java, i.e. JDK1.7.0 version.

The Net-Beans simulation for proposed system is as shown in Fig. 2. This simulation consists of a Master node, a Job Request module and three Local Node Managers. The important component, i.e. a Phase-Based Scheduler is appeared in Master node. This Phase-Based Scheduler is responsible for scheduling the phases and allocating the resources. This scheduler is also known as a Fine-Grained Resource-Aware Scheduler because; it allocates the resources as per the demand of the phases, (i.e. neither overflow nor underflow).

Another vital component of Master node is Job Progress Monitor. This is responsible to capture the phase-level progress information, i.e. a particular phase is completed or not.

Next is the Job Request module. In this module, a user can browse a file which is to be mapped and reduced and then, this file is assigned to the Job Monitor for further execution.

Finally, there are three Local Node Managers, which are responsible for coordinating transitions between phase and the scheduler.

In this manner, the Phase-Level scheduling, i.e. proposed system of PRISM is implemented. After successful completion of all three jobs in the Master node; certain graphs are obtained as results, which are explained in the next sub-section.

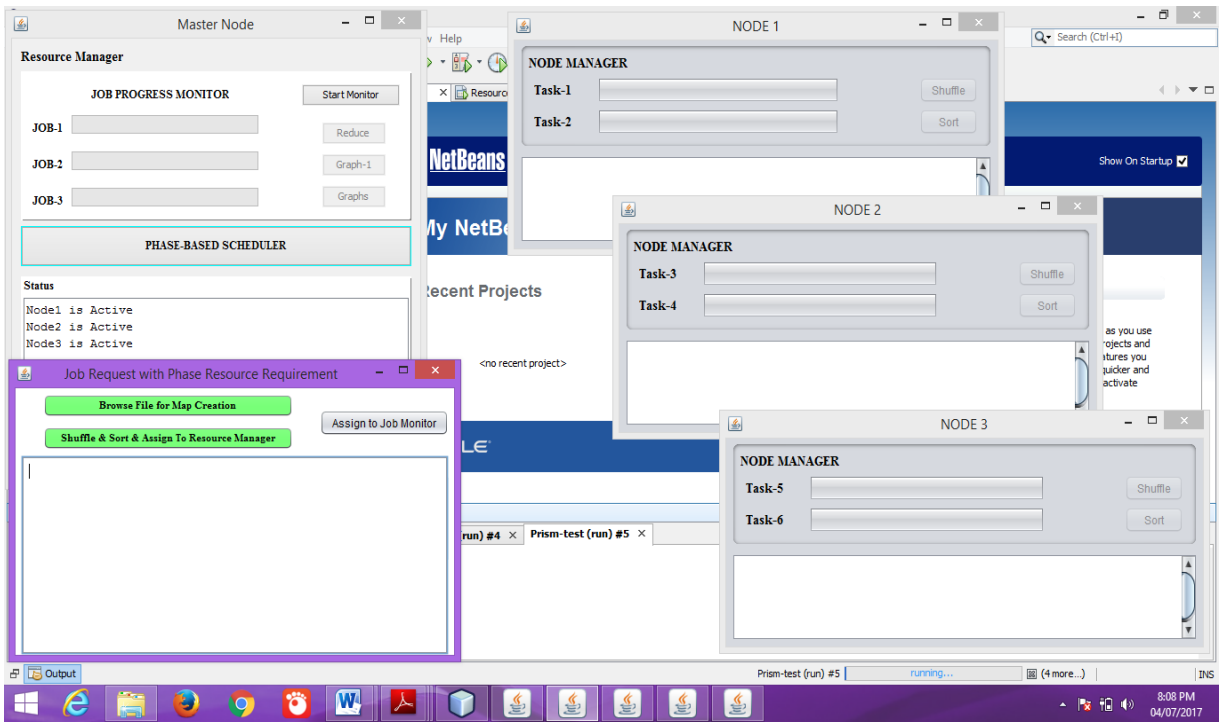

Fig. 2: Simulation of PRISM

5.2 Experimental Results

As it is discussed that, PRISM architecture is implemented as both, i.e. existing and proposed systems; therefore certain graphs are obtained as results in both the systems.

The existing system is based upon Task-Level scheduling; whereas the proposed system is entirely based upon Phase-Level scheduling.

5.2.1 Existing System and its Results

In the Existing system (i.e. Task-level Scheduling), it is difficult for schedulers to effectively utilize the available resources in order to reduce job execution time, because tasks may have highly varying resource requirements during their lifetime. Also, there is no pipelining of tasks (i.e. subsequent phases are not scheduled simultaneously).

As a result, there is a huge delay problem. Hence, existing system is too much time-consuming system. Due to this, the performance and efficiency of the system is reduced.

The following Fig. 3 shows how the tasks suffer from delay problem. As shown in Fig. 3, there is a XY Line Chart example, in which there are three nodes (Node 1, Node 3, and Node 5) on X-axis and there is Time (in sec) on Y-axis.

It can be observed from this graph structure that, each and every node consumes too much time to complete its execution. Because of this delay in execution, the existing system slows down its performance.

In short, these are the drawbacks of existing system, which are minimum utilization of resources, no pipelining, delay problem and insufficient scheduler decision making.

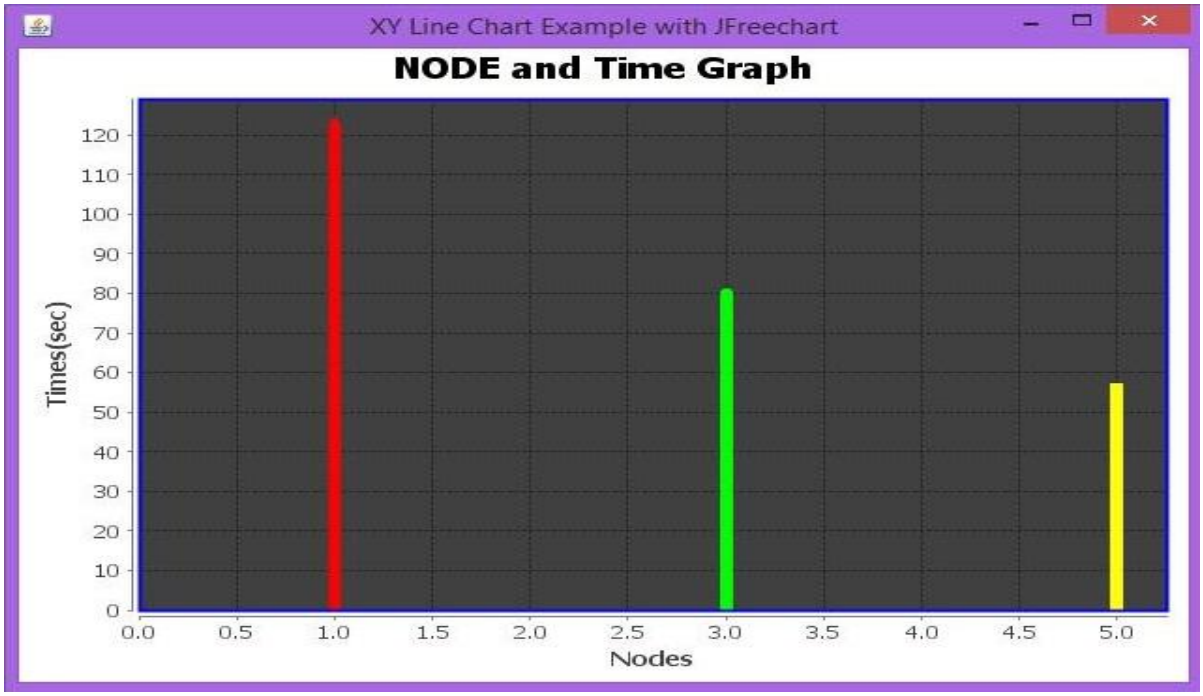

Fig. 3: Node and Time Graph (existing system)

Additionally, there are two results (i.e. Line charts). In existing system, map and reduce functions are implemented. The first Line chart describes the CPU and memory usages required for executing map and reduce functions (i.e. Fig. 4).

In this Line chart, the X-axis shows how much time (in sec) the CPU and Memory has taken during execution and the Y-axis shows how much percentage of CPU and Memory has been utilized.

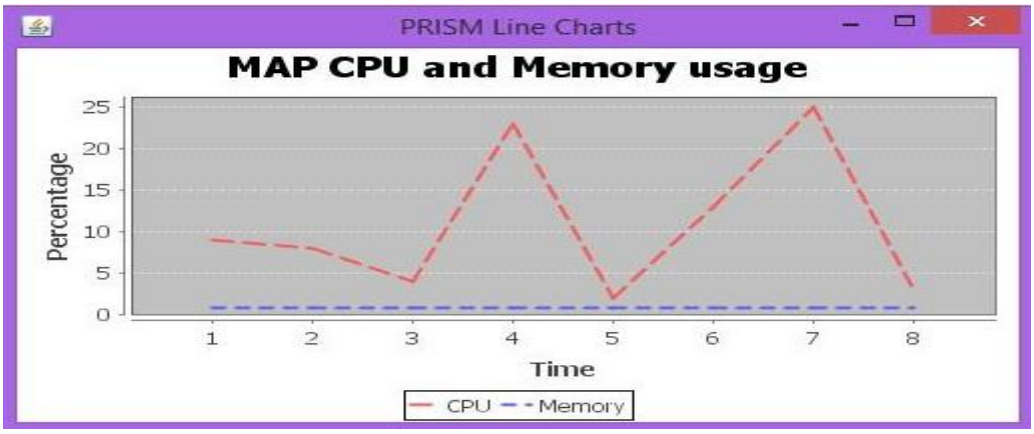

Fig. 4: Map CPU and Memory Usage (existing system)

Next Line chart is about Map IO usage (i.e. Fig. 5). In this Line chart, the X-axis shows how much time (in sec) the Local Disk I/O and HDFS I/O has taken during execution and Y-axis describes the rate (MB/sec) of data for which the I/O devices has been used.

These I/O devices are used as buffers for storing data blocks while running map-reduce functions.

During the execution, the data blocks are first stored on HDFS (i.e. Hadoop Distributed File System) for implementing map-reduce functions. When this buffer becomes full, then the remaining content is written into the Local Disk I/O buffer.

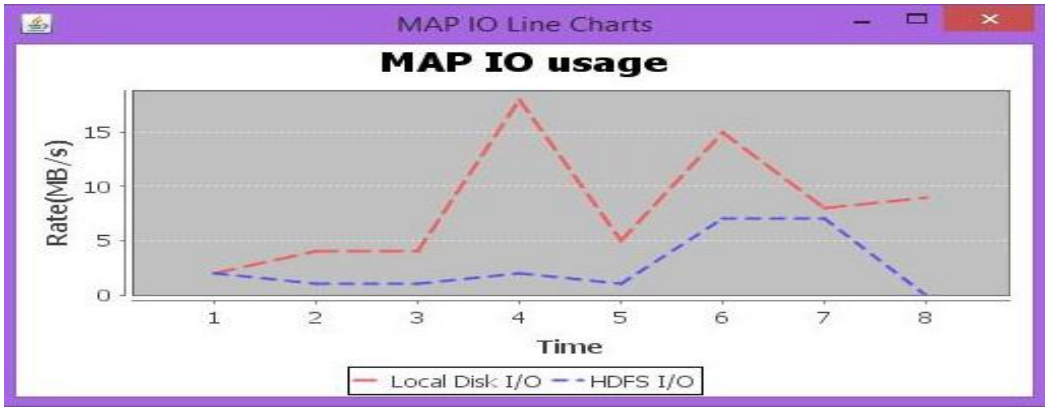

Fig. 5: Map I/O Usage (existing system)

5.2.2 Proposed System and its Results

The Proposed System (i.e. Phase-level Scheduling) introduces PRISM, which is a fine-grained resource-aware map-Reduce scheduler. PRISM divides tasks into phases, where each phase has a persistent resource usage profile and implements scheduling at the level of phases.

In the proposed system, schedulers allocate the resources as per the demand of the phases during run-time. Here, the concept of pipelining is used (i.e. subsequent phases are scheduled simultaneously), which avoids resource contention and enhances resource utilization. Therefore, there is no time consumption, which improves the speed of job Running Time. Overall, proposed system achieves high job Performance and efficiency.

The following Fig. 6 shows the Node and Time Graph of proposed system. As shown in this figure, there are three nodes (Node 1, Node 3, and Node 5) on X-axis and Time (in sec) on Y-axis.

Because of pipelining, it can be observed from this graph structure that, each and every node consumes as minimum time as needed for execution. Due to this, the performance and execution speed of the proposed system is appreciably enhanced.

In short, these are the advantages of proposed system, which are maximum utilization of resources, pipelining (improvement in job running time), no delay problem and achievement in high job performance.

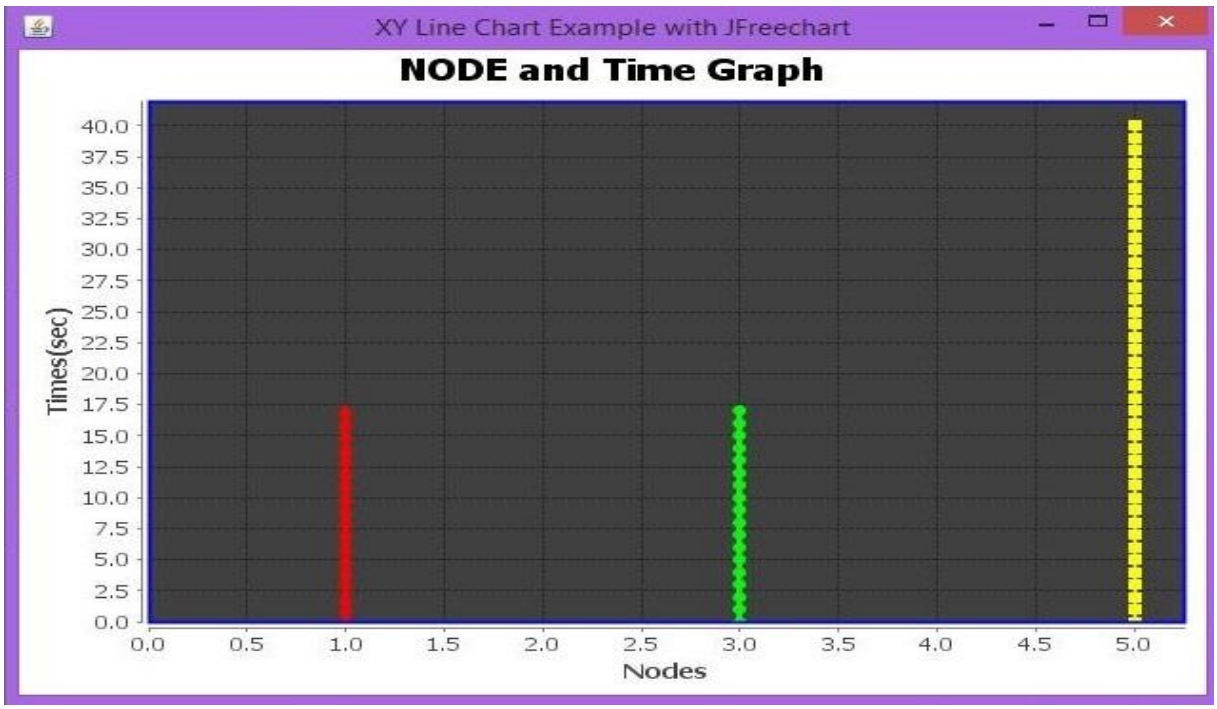

Fig. 6: Node and Time Graph (proposed system)

Moreover, there are two results (i.e. Line charts) in the proposed system too. In this system, shuffle, sort and reduce functions are implemented. The first Line chart describes the CPU and memory usages required for executing the shuffle, sort and reduce functions (i.e. Fig. 7).

In this Line chart, the X-axis indicates how much time (in sec) the CPU and Memory has taken during the execution and the Y-axis describes how much percentage of CPU and Memory has been utilized.

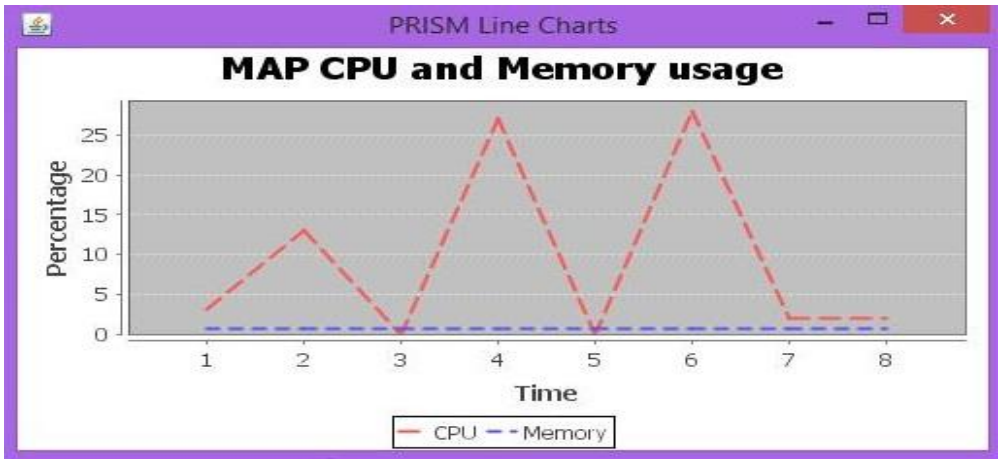

Fig. 7: Map CPU and Memory Usage (proposed system)

At the end of execution, the final Line chart is about Map IO usage (i.e. Fig. 8). In this Line chart, the X-axis indicates how much time (in sec) the Local Disk I/O and HDFS I/O had taken during execution and Y-axis defines the rate (MB/sec) of data blocks for which the I/O devices has been used.

These I/O devices are referred to as buffers for storing data blocks while running the shuffle, sort and reduce functions.

During the execution, the data blocks are first stored on HDFS buffer (i.e. Hadoop Distributed File System) for implementing the specified functions. When this buffer is fully occupied, then the remaining content is written into the Local Disk I/O buffer. Hence, these are the Graphs and Line charts, which appear while executing the proposed system.

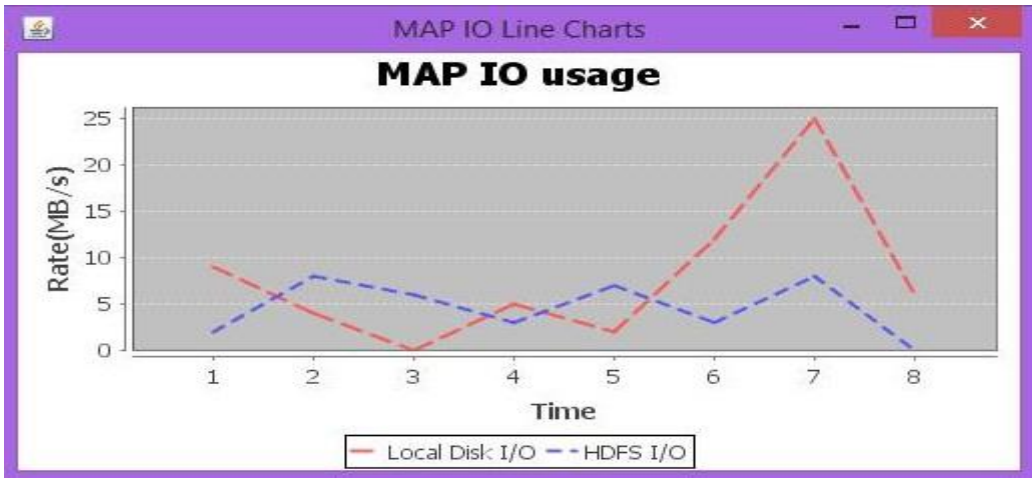

Fig. 8: Map I/O Usage (proposed system)

6. CONCLUSION AND FUTURE SCOPE

Therefore, in this paper, Map-Reduce is used a popular programming model for computing the data intensive jobs. PRISM, i.e., a fine-grained resource-aware Map-Reduce scheduler, divides tasks into phases and also performs scheduling at the phase level. Because of using this Phase-level scheduling, there is enhancement in Resource Utilization.

The scheduling algorithm used by PRISM contributes in minimization of job running time as compared to the current Hadoop schedulers. Overall, PRISM achieves high job Performance. Finally, the future scope of this paper will be improvement in the scalability of PRISM by using the distributed schedulers.

7. REFERENCES

[1] Hadoop MapReduce distribution [Online]. Available: http://hadoop.apache.org, 2015.

[2] Hadoop Capacity Scheduler [Online]. Available: http://hadoop.apache.org/docs/stable/capacity_scheduler html/, 2015.

[3] Hadoop Fair Scheduler [Online]. Available: http://hadoop.apache.org/docs/r0.20.2/fair_scheduler.html, 2015.

[4] Hadoop Distributed File System [Online]. Available: hadoop.apache.org/docs/hdfs/current/, 2015.

[5] GridMix benchmark for Hadoop clusters [Online]. Available:http://hadoop.apache.org/docs/mapreduce/curt/gridmix.html, 2015.

[6] PUMA benchmarks [Online]. Available: http://web.ics.purdue.edu/fahmad/benchmarks/datasets.htm, 2015.

[7] The Next Generation of Apache Hadoop MapReduce [Online].Available:http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, 2015.

[8] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," in Proc. USENIX Symp. Netw. Syst. Des. Implementation, 2010, p. 21.

[9] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.

[10] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in Proc. USENIX Symp. Netw. Syst. Des. Implementation, 2011, pp. 323–336.

[11] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in Proc. Conf. Innovative Data Syst. Res., 2011, pp. 261–272.

[12] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, and K. Talwar, "Quincy: Fair scheduling for distributed computing clusters," in Proc. ACMSIGOPS Symp. Oper. Syst. Principles, 2009, pp. 261–276.

[13] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. "Multi-resource allocation: Flexible tradeoffs in a unifying framework," in Proc. IEEE Int. Conf. Comput. Commun., 2012, pp. 1206–1214.

[14] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguad_e, "Resource-aware adaptive scheduling for MapReduce clusters," in Proc. ACM/IFIP/USENIX Int. Conf. Middleware, 2011, pp. 187–207.

[15] Verma, L. Cherkasova, and R. Campbell, "Resource provisioning framework for MapReduce jobs with performance goals," in Proc. ACM/IFIP/USENIX Int. Conf. Middleware, 2011, pp. 165–186.

[16] Qi Zhang, Student Member, IEEE, Mohamed Faten Zhani, Member, IEEE, Yuke Yang, Raouf Boutaba, Fellow, IEEE, and Bernard Wong, "PRISM: Fine-Grained Resource-Aware Scheduling for Map-Reduce," in ieee transactions on cloud computing, vol. 3, no. 2, april/june 2015.