International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)
ISSN: 0976-1353 Volume 22 Issue 1 – MAY 2016.

# PROCESSOR LEVEL RESOURCE-AWARE SCHEDULING FOR MAPREDUCE IN HADOOP

G.Hemalatha [#1] and S.Shibia Malar [*2]

[#] *P.G Student, Department of Computer Science, Thirumalai College of Engineering, Kanchipuram, India*

[*] *Assistant Professor, Department of Computer Science, Thirumalai College of Engineering, Kanchipuram,India*

***Abstract*— MapReduce is an effective model that aids the programmers to run their application over hundreds, thousands, or even tens of thousands of machines in a cluster. By breaking down each job into small map and reduce tasks and executing them in parallel across a number of machines, MapReduce significantly reduces the running time of data-intensive jobs. However, despite recent effort towards designing resource-efficient MapReduce schedulers, existing solutions that focus on scheduling at the task and phase-level still offer sub-optimal job performance. This paper proposes a processor-level scheduling that improves job running time and high resource utilization without introducing stragglers.**

***Index Terms*—Hadoop, MapReduce, Resource allocation, Stragglers.**

## I. Introduction

In recent years large scale data industry is trending towards Data-intensive computing. MapReduce that breaks down job into map and reduce tasks are scheduled parallel across multiple machines. These tasks significantly reduces job running time. Subsequently, the performance and efficiency for scheduling the job under various resource requirements provide resource utilization and support today’s internet companies.

The key component of MapReduce is its Job Scheduler. The scheduler schedules map and reduce tasks which reduces job running time and increases resource utilization. A schedule with large numbers of tasks running on a single machine in resource contention and the machine with very few tasks will lead to poor resource utilization.

The job scheduling is quite easier in systems having homogenous resource requirements in terms of CPU, memory, Network bandwidth. In PRISM, a Phase and Resource Information-aware Scheduler for MapReduce clusters, the running time of the tasks vary significantly from phase. Consequently, the resource demand at the phase level is possible for the scheduler to achieve higher degrees of parallelism avoiding resource contention. To this end, a phase-level scheduling algorithm delivers up to 18 percent improvement in resource utilization while allowing jobs to complete up to 3 times faster than current Hadoop schedulers. To end with PRISM is currently designed for Hadoop MapReduce, it can be functional to Dryad [19] and other parallel computing frameworks as well.

Recent studies have reported that workloads often have assorted utilization profiles and performance requirements [8], [20]. Failing to consider these job usage features can possibly lead to inept job schedules with low resource utilization and long job execution time. Certainly, current MapReduce systems, such as Hadoop MapReduce schedulers make this assumption to abridge the scheduling problem. These systems use a phase-level scheduling algorithm, where the physical resources on each machine are captured by the assigned to tasks and phase-level. Unfortunately, in practice, running-time of resource consumption varies from phase to phase and from job to job.

Motivated by this observation, some recent proposals, such as PRISM, a Phase and Resource Information-aware Scheduler for MapReduce resource-aware adaptive scheduling [15] and Hadoop MapReduce Version 2 [7], have presented resource aware job schedulers to the MapReduce framework. Clusters that performs resource-aware scheduling at the level of task phases. Precisely, for most MapReduce applications, resource consumption varies significantly from phase to phase. Therefore, the resource demand of the scheduler to reaches higher degrees of parallelism avoiding resource contention.

This paper is structured as follows. Section II covers an overview of Hadoop MapReduce and PRISM. Section III provides the Processor-Level scheduling. The algorithm is described in section IV. The performance evaluation is presented in section V. The related work of the existing system is summarized in section VI. Conclusion and Future enhancement are cited in Section VII and VIII respectively followed by Reference.

## II. Background

### A. *Hadoop MapReduce*

A parallel computing model known as MapReduce is widely used for large scale data intensive computation. In MapReduce, a job entails map and reduced tasks. A map task takes a key-value block as input which is stored in the core distributed file system. A user specified map function is run to generate intermediary key-value output. Consequently, a reduce task collects and apply the user-specified reduce function to generate the output. Apache Hadoop MapReduce is the software framework widely used to implement MapReduce. The Hadoop cluster is comprised of large number of commodity machines where single node serves as a master and other acts as a slaves. The master node which is resource manager Job Tracker programs job to the slave nodes. For each task a Local node manager or task tracker launches and allocates resources with the help of Java Virtual Machine. In earlier versions, Hadoop MapReduce accomplishes a slot based resource allocation scheme.

A Hadoop cluster is a multi-user system where many users submit jobs to the cluster at the same time. The Resource Manager upholds a job list. The progress of the running task and the available resources in the node are monitored by the respective slave node. Periodically the slave nodes sends a heartbeat message and transfer information to the master node. With the provided information the resource scheduler makes the scheduling decision Hadoop MapReduce also supports other task level job schedulers such as Capacity Scheduler [2] and Fair Scheduler [3]. As task-level scheduling implements a simple slot based allocation run-time task is not considered here. This leads to resource contention if multiple tasks are assigned.

From this observation, Resource-Level scheduling in Hadoop MapReduce clusters is facilitated by Hadoop Yarn. In alpha version, the size of the task container for the task process can be specified.

### B. PRISM

PRISM or Phase and Resource Information-aware scheduler for MapReduce clusters performs resource-aware scheduling at the phase level avoids resource contention. An overview of the PRISM mechanism is shown in Fig. 1. PRISM consists of three main components:

- Phase-Based Scheduler at the master node
- Local Node Managers that coordinate phase transitions with the scheduler
- Job Progress Monitor to capture phase-level progress information.

In the Phase-Level scheduling mechanism, (1) the Node Manager transmits a heartbeat message to the scheduler. (2) The scheduler receives the heartbeat message and replies to the request. (3) The task is then launched by the Node Manager. (4) Once the task completes the execution in a particular phase it asks for permission to begin the next phase. The scheduler receives the permission request from the Local Node Manager. (5) Based on the resource requirements and the current progress information, the scheduler makes scheduling decision. The scheduler either starts a new task or pauses a task and begin the task in next phase and informs to the Node Manager. (6) If the task is executed in the next phase the Node Manager provides permission to complete the task process. (7) The Node Manager receives task status (8) and forwards task status to the scheduler

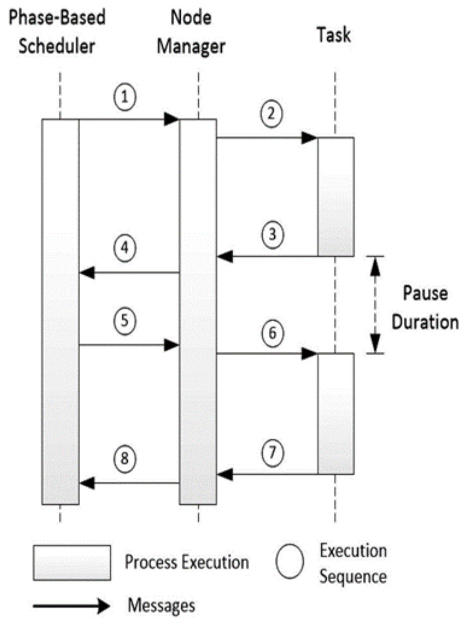

Fig. 1 PRISM Mechanism

From this it is observed that by pausing the task at run-time results time delay for job completion of the current and subsequent tasks which leads to stragglers.

## III. PROCESSOR LEVEL SCHEDULING

The importance of processor-level scheduling is demonstrated in this paper. In a phase-level, performance of a task or process with heterogeneous resource requirements is carried out. In processor-level scheduling algorithm execution parallelism and performance of task is improved. PRISM, i.e. Phase and Resource Information-aware Scheduler for MapReduce at the phase-level is an existing solution. While preceding a task, it has many run-time resources within its lifetime. While scheduling the job, PRISM always have higher degree of parallelism than current Hadoop cluster. It refers at the phase-level to improve resource utilization and performance. In this the thread will assign jobs to the processor, but the thread doesn't know about the processor. After the jobs are assigned to the processor, information about the job size will go and the map reduces and then MapReduce know the processor capacity and it assign the job based on their Processor capacity.

The processor will complete all works that are assigned does not leave any unfinished jobs as like the existing ones. After that the results of each processor will go to the i2FAR Mechanism. In this Mechanism it checks if all the jobs are finished by the processor it will go to Success Result Otherwise If any jobs are incomplete it will go to the map reduce and it will find the processors and make that processor to complete the jobs.

In distributed system while distributing the tasks the thread allow task to the processor. But they don't know about the processor capability. Suppose if the thread are allowing a very

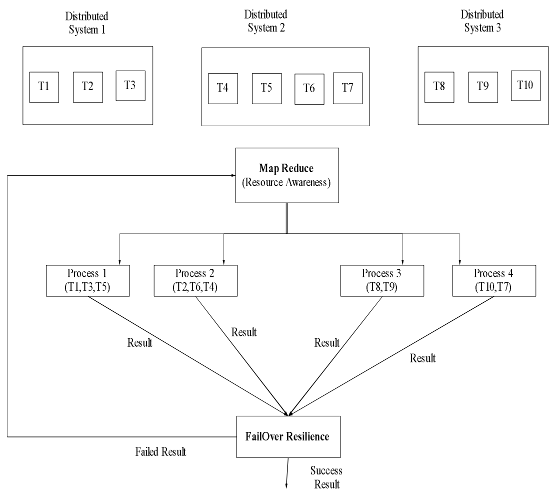

Fig. 2 Processor level Scheduling.

huge task to the processor without knowing their capacity. The processor will do their task according to their capability. But the remaining works are left undone. Here the distributed system will processed in to the thread in that the system knows about the capacity of the thread. Based on the thread the process will be synchronized.

While allocating task with help of MapReduce they know their capability of each processor. When the tasks are allocated to the processor they will go the Map Reduce and they will assign the task to the processor by knowing their capability. Then thread will assign the task to the processor and it will send it to the MapReduce and then the MapReduce will send the task to the Processor, it will finished the task and send the result to the I2 FAR. If any processors works are left undone it will send to the processor again and complete the left work

I2 FAR will give the results if the processor finished their jobs it will give the results as success. If the jobs are left undone it will go to the processor again complete their tasks. The received result will finished then it will go to the i2FAR process and produced to execute complete process. If the result is not completed then it will be re forwarded to the reduced map mechanism. There the remaining uncompleted results will be completed.

## IV. Algorithm Description

As soon as the processor finishes their job it offers a success result. In case of any incomplete jobs that is if any of the job is not done by the processor. It is forwarded to the MapReduce Mechanism again. Here the jobs are allocated to the processor that are free. The processor that completes the job will go to that i2FAR to execute the complete process. Considering J jobs are assigned to the Processor P in Algorithm 1. For each job j assigned to the processor p. The phase selected for the assigned job has the utility function U {j}. If job j is not done in the phase selected it is assigned to the processor that finishes the job. U {j} is assigned to the processor in the selected phase. The resource utilization is updated until all the jobs are assigned to the processor.

### A. *i2FAR ALGORITHM*

```
1. Input from the users
2. Obtain the Input from the users
3. Inputs are allocated to each processor.
4. Each Job is assigned to the processor j € p
5. Repeat
6. for each processor the job is assigned
     P🡐J
     endfor
  7. if processor = job assigned
   8 phase selected🡐phase selected U{j}
   9. endif
10. Endfor
11. if processor ≠ job assigned
12.PhaseSelected🡐PhaseSelected U{j}
13. p 🡐select the processor that has finished the job
14. Update the resource utilization of the machine
15. end if
16. endif
17. until p==job assigned
18. return phase
```

Supposing each machine run at most N tasks, the scheduler considers N schedulable processor for each job. The ranking of the processor is carried out by the local manager. Thus the overall running time of the Job is calculated using this algorithm. Though the scheduling will not improve phase-level and task level parallelism, the resources are shared thus avoiding stragglers. It should be said that the processor level scheduling assigns job to the processor in case of any jobs left undone. The scheduler then reassign the job to the processor that are free and update the resource utilization reducing the job failures.

## V. Performance Evaluation

In this processor –level scheduling, a cluster consisting of two nodes is run using Apache Hadoop 2.7.2. The input of size 4 GB consisting of whether data of various countries are used. The input data are divided into Map and Reduce tasks. The jobs are assigned to the processor and are executed without stragglers using failover resilience Mechanism. This mechanism checks whether the processor finishes assigned undone jobs and completes the job execution without stragglers.

The result is compared with the PRISM and as expected the job is completed with high resource utilization without introducing stragglers. And the job running time is less when compared to the existing schedulers.

For better understanding we considered the resource such as CPU, Memory Usage, and Network I/O usage. With the previous results the yarn and Fair scheduler are worse than

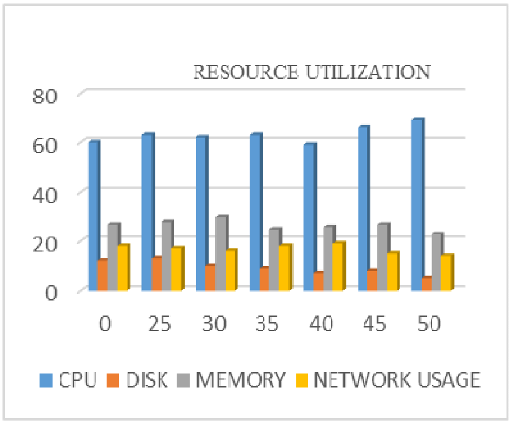

Fig.3 Resource Utilization in Processor-Level Scheduling.

PRISM. While PRISM introduces stragglers and this has less resource utilization and job completion time when compared to processor-level scheduling algorithm.

Fig.3. Resource utilization of various resource requirements such as CPU, Network, Disk, memory usage are plotted. The performance gain of the processor-level scheduling is higher than the phase-level scheduling. Hadoop achieves higher resource utilization than the current schedulers such as yarn and Fair scheduler. The Application Master that monitors and manages the job execution also consumes lesser running time.

The Map, Reduce and Shuffle phases reduce the running time as the can run simultaneously. The time delay in the reduced phase is scheduled by scheduling the shuffle phases by knowing the processor capacity. Therefore achieves high resource utilization with reduced resource contention.

The running time of job various job with different input size are given and they are compared with the existing phase-level scheduling. The input size for various job in processor-level scheduling offers a less job running time when compared to the existing scheduling mechanism. The running time for the phase-level and processor level for various job with different input size are plotted in Fig.4

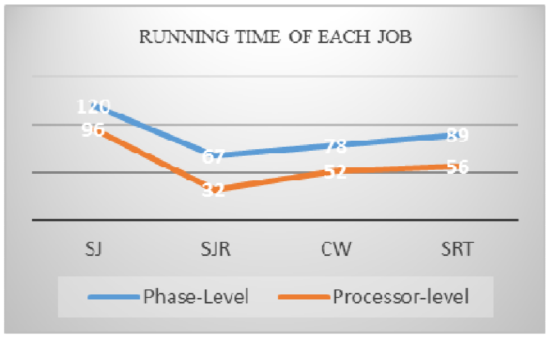

Fig.4 Running Time of Job Compared to Phase-Level Scheduling

## VI. RELATED WORK

MapReduce application runs on a large cluster of commodity machines and is highly scalable. A typical MapReduce computation processes a very large data on thousands of machines. Many MapReduce programs have been executed on Google's clusters every day. Hadoop MapReduce implements a slot-based resource allocation scheme, which does not consider run-time task resource consumption. As a result, several recent works reported the inefficiency introduced due to such simple design, and proposed solutions. For instance, Polo et al. proposed RAS [15], an adaptive resource-aware scheduler that uses a resource-aware scheduling technique for MapReduce multi-job workloads that aims to improve resource utilization across machines observes completion time goals. However, RAS still performs scheduling at task-level, and does not consider the task resource usage variations at run time. Subsequently, Hadoop Yarn [7] represents a major endeavors in resource-aware scheduling in MapReduce clusters. It offers the ability to specify the size of each task container in terms of requirements for each type of resources. In this context, a key challenge is to define the notion of fairness when multiple resource types are considered. Ghodsi et al. proposed dominant resource fairness as a measure of fairness in the presence of multiple resource types, and provided a simple scheduling algorithm for achieving near-optimal DRF. However, the DRF scheduling algorithm still focuses on task-level scheduling, and does not consider change in resource consumption within individual tasks. Their subsequent model, namely. Dominant resource fair queueing (DRFQ), aims at achieving DRF for packet scheduling over time. However, DRFQ algorithm is mainly designed for packet scheduling, which is different from the task-level “bin-packing” type of scheduling model we consider in this paper. Thus it cannot be directly applied to MapReduce scheduling. Using profiles to improve MapReduce job performance has received considerable attention in recent years [12]. For instance, Verma et al. [17] developed a framework that profiles task running times and use the job profiles to achieve deadline-ware scheduling in MapReduce clusters. Herodotou et al. recently developed Starfish [12], a job profiler that collects fine-grained task usage characteristics that can be used for fine-tuning job configuration parameters. However, the goal of profiling in these studies is to optimize job parameters, rather than optimizing job schedules. Another related research direction is MapReduce pipelining. In particular, MapReduce Online [9] is a framework for stream-based processing of MapReduce jobs. It allows partial outputs of each phase to be sent directly to the subsequent phase, thus enables overlaps execution of phases. In order to minimize I/O, ThemisisMR [16] is another scheme that makes fundamentally different design decisions from previous MapReduce implementations. Themis performs an extensive variety of MapReduce jobs at nearly the speed of Triton Sort's record-setting sort performance. However, both of these solutions does not deal with scheduling. Furthermore, they are not resource-aware. While introducing resource awareness in MapReduce Online is another interesting alternative, the scheduling model for MapReduce online is much different from the current MapReduce. It will require further investigation to identify scheduling issues for Map- Reduce online.

## VII. CONCLUSION

In this paper we assume all machines have identical hardware and resource capacity. It is oblivious to the fact that the execution of each task can be divided into phases with drastically different resource consumption characteristics. To address this limitation, PRISM, a fine-grained resource-aware scheduler that coordinates task execution at the level of phase. In the existing System the thread doesn't know about processor capability it assign a huge task to the processor , So the processor will do their tasks according to their capability, The remaining left jobs are not done. In the Recommended System the Map reduce will get the task from the thread and it assign the task to the processor according to their capability with this all the assigned task are done.

## VIII. FUTURE ENHANCEMENT

It is interesting to study the profiling and scheduling problem for machines with heterogeneous performance characteristics. Finally, improving the scalability of PRISM using distributed schedulers is also an interesting direction for future research.

## REFERENCES

[1] Hadoop Map Reduce 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] Grid Mix benchmark for Hadoop clusters [Online]. Available: http://hadoop.apache.org/docs/mapreduce/current/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] R. Boutaba, L. Cheng, and Q. Zhang, "On cloud computational models and the heterogeneity challenge," J. Internet Serv. Appl., vol. 3, no. 1, pp. 1–10, 2012.
[9] 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.
[10] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.
[11] 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.
[12] 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.
[13] 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.
[14] 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.
[15] 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.
[16] A. Rasmussen, M. Conley, R. Kapoor, V. T. Lam, G. Porter, and A. Vahdat, "ThemisMR: An I/O-Efficient MapReduce," in Proc. ACM Symp. Cloud Compute. 2012, p. 13.
[17] A. 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.
[18] D. Xie, N. Ding, Y. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in Proc. ACM SIGCOMM, 2012, pp. 199–210.
[19] Y. Yu, M. Isard, D. Fetterly, M. Budiu, _ U. Erlingsson, P. Gunda, and J. Currey, "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language," in Proc. USENIX Symp. Oper. Syst. Des. Implementation, 2008, pp. 1–14.
[20] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in Proc. Eur. Conf. Comput. Syst., 2010, pp. 265–278.
[21] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in Proc. USENIX Symp. Oper. Syst. Des. Implementation, 2008, vol. 8, pp. 29–42.