ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

Efficient Job Execution for Map Reduce Using Phase-Level Scheduling Algorithm

Mary Vennila S, Prabhakarana S

PG Student, Dept. of Computer Science and Engineering, Muthayammal Engineering College, Rasipuram, Namakkal, Tamilnadu, India

Professor, Muthayammal Engineering College, Rasipuram, Namakkal, Tamilnadu, India

ABSTRACT: Map Reduce has become a popular model for data-intensive computation in recent years. It can significantly reduce the running time of data-intensive jobs by breaking down each job into small map and reduce tasks and executing them in parallel across a large number of machines. Designing resource-efficient Map Reduce schedulers is important for effectively reducing the job running time. Existing solutions are focused on scheduling a job at task-level. And the tasks can have highly varying resource requirements during their lifetime, which makes it difficult for task-level schedulers to effectively utilize available resources to reduce job execution time. To address this limitation, in existing system they introduced a scheduler called PRISM, a fine-grained resource-aware Map Reduce scheduler that divides tasks into phases, where each phase has a constant resource usage profile, and performs scheduling at the phase level. They first demonstrate the importance of phase-level scheduling by showing the resource usage variability within the lifetime of a task using a wide-range of Map Reduce jobs. And then present a phase-level scheduling algorithm that improves execution parallelism and resource utilization without introducing stragglers. There is a limitation in PRISM scheduling, such that when allocating each phase into the scheduler the phase will execute when only the resources available. If not, the phase will be paused until getting the resource available and the next phase is scheduled to execute. Due to the pausing the phase will make delay in running a job. In my proposed system to address this limitation I will use a concept of virtual resource allocation, instead of pausing the phase it will be availed to use the resources virtually until getting the resource available. It will reduce the delay in running a job significantly.

KEYWORDS: Data intensive computation, MapReduce, scheduling, resource allocation

1. INTRODUCTION

Data-intensive computation is a class of parallel computing applications which use a data parallel approach to process large volume of data typically terabytes or petabytes in size and typically referred to as big data.

In MapReduce, a job is a collection of map and reduce tasks that can be scheduled concurrently on multiple machines, resulting in significant reduction in job running time. Many large companies, such as Google, routinely use MapReduce to process large volumes of data on a daily basis. Consequently, the performance and efficiency of MapReduce frameworks have become critical to the success of today's internet companies. Job scheduler is a central component of a MapReduce system. Its role is to create a schedule of Map and Reduce tasks, spanning one or more jobs, to minimize job completion time and maximizes resource utilization. A schedule with too many concurrently running tasks on a single machine will result in heavy resource contention and long job completion time. Conversely,

a schedule with too few concurrently running tasks on a single machine will cause the machine to have poor resource utilization. , current MapReduce systems, such as Hadoop MapReduce Version.

These systems use a simple slot-based resource allocation scheme, where physical resources on each machine are captured by the number of identical slots that can be assigned to tasks. Run-time resource consumption varies from task to task and from job to job. Several recent studies have reported that production workloads often have diverse utilization profiles and performance requirements. Failing to consider these job usage characteristics can potentially lead to inefficient job schedules with low resource utilization and long job execution time. Motivated by this observation, several recent proposals, such as resource-aware adaptive scheduling (RAS) and Hadoop MapReduce

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

Version 2 (also known as Hadoop NextGen and Hadoop Yarn) [7], have introduced resource aware job schedulers to the MapReduce framework.In this paper, we include

PRISM, a Phase and Resource Information-aware Scheduler for MapReduce clusters that performs resource-aware scheduling at the level of task into phases. Specifically, we show that for most MapReduce applications, the run-time task resource consumption can vary significantly from phase to phase. Therefore, by considering the resource demand at the phase level, it is possible for the scheduler to achieve higher degrees of parallelism while avoiding resource contention. To this end, we have developed a phase-level scheduling algorithm with the aim of achieving high job performance and resource utilization.Through experiments using a real MapReduce cluster running a wide-range of workloads, we show PRISM delivers up to 18 percent improvement in resource utilization while allowing jobs to complete up to 1:3 faster than current Hadoop schedulers.

The rest of the paper is organized as sections. Section 2 provides a basic overview of MapReduce scheduling and job execution. In Section 3 we describe the phase-level task usage characteristics and our motivation. Section 4 introduces PRISM and describes its architecture. The phase-level scheduling algorithm is presented in detailsin Section 5. Our experimental evaluation of PRISM is provided in Section 6. Finally, we summarize existing work related to PRISM in Section 7, and draw our conclusion in Section 8

II. BACKGROUND

2.1 MAPREDUCE AND HADOOP

The execution of a MapReduce job is divided into a Map phase and a Reduce phase. In the Map phase, the map tasks of the job are executed. Each map task comprises the execution of the job's map() function as well as some supporting actions for example, data sorting. The data output by each map task is written into a circular memory buffer when this buffer reaches a threshold, its content is sorted by key and moved to a temporary file.

These files are then served to reduce phase. Reduce phase is divided into three sub-phases as shuffle, sort and reduce. The shuffle sub phase copies the map output from the map machines to the reducer's machine. The sort sub phase is responsible for sorting the intermediate data by key. The last reduce sub phase runs the job's reduce() function, and the final result is written to the distributed file system.

Hadoop is an open source implementation of MapReduce provided by Apache Software Foundation. The Hadoop architecture follows the master/slave paradigm, consists of a master machine responsible for coordinating the distribution of work and execution of jobs, and a set of worker machine responsible for performing work assigned by the master. The master and slaves roles are performed by the 'JobTracker' and 'TaskTracker' processes. The singleton JobTracker partitions the input data into 'input splits' using a splitting method defined by the programmer, populates a local task-queue based on the number of obtained input splits, and distributes work to the TaskTrackers that in turn process individual splits. Work units are represented by 'tasks' in this framework. There is one map task for every input split generated by the JobTracker. The number of reduce tasks is defined by the user and each TaskTracker controls the execution of the tasks assigned to its hosting machine.

.2 PHASE-LEVEL SCHEDULING

This section deals with evaluating the phase-level resource requirements across various jobs, including the examples provided by the Hadoop MapReduce distribution. These observations suggest that the runtime task resource consumption is dependent on the phase in which the task is currently executing. Therefore, ignoring the phase-level resource characteristics will lead to poor resource allocations decisions, which in turn will cause the job scheduler to make inefficient job scheduling decisions.

Hence, the phases of incoming job is further divided into more fine-grained phases to achieve even more uniform resource usage in each phase, we found that doing this is cost prohibitive because (1)certain fine-grained phases (e.g. partition and spill [12]) are tightly coupled with each other thus scheduling them individually will require major change to the MapReduce implementation, and (2) a more fine-grained splitting as this will significantly increase the complexity of the system and the scheduling overhead may outweigh the gain attained by phase-level scheduling.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

III. PRISM

Allocating a fixed sized container for each task can lead to inefficient scheduling decisions. At run-time, if the resource allocated to the task is higher than the current resource usage, then idle resources are wasted. Or in contrary, if the resource allocated to the task is much less than the actual task resource demand, the resource can become a performance bottleneck and slow down task execution. This motivates us to design a fine grained, phase-level scheduling scheme that allocates resources according to the phase that each task is currently executing. By exploiting fine-grained phase-level resource characteristics, it is possible to achieve higher resource utilization compared to task-level schedulers.

A key issue that must be addressed in phase-level scheduling is that once a task has completed a phase, the next phase of the task may not be scheduled immediately if the machine does not have sufficient resources to run the subsequent phase. Thus, the execution of a phase may be "paused" in order to avoid resource contention, at the cost of delaying the completion of the task.

In existing paper, they present PRISM, a finegrained resource-aware scheduler that performs scheduling at phase-level. Unlike existing MapReduce schedulers that only allow job owners to specify resource requirements at task-level, PRISM allows the job owners to specify phaselevel resource requirements. An overview of the PRISM architecture is shown in Fig.1.

PRISM consists of three main components: a phase-based scheduler at the master node, local node managers that coordinate phase transitions with the scheduler, and a job progress monitor to capture phase-level progress information. The phase-level scheduling mechanism used by PRISM is illustrated by Fig.2. Hadoop implementation, each node manager periodically sends a heartbeat message to the scheduler. when a task needs to be scheduled, the scheduler replies to the heartbeat message with a task scheduling request (Step 1). The node manager then launches the task (Step 2). Each time a task finishes executing a particular phase (e.g. shuffle phase of the reduce task), the task asks the node manager for a permission to start the next phase (e.g. reduce phase of the task) (Step 3).

The local node manager then forwards the permission request to the scheduler through the regular heartbeat message (Step 4). Given a job's phase-level resource requirements and its current progress information, the schedulerdecides whether to start a new task, or allow a paused task to begin its next phase (e.g., the reduce phase), and then informs the node manager about the scheduling decision (Step 5). Finally, once the task is allowed to execute the next phase, the node manager grants the permission to the task process (Step 6). Once the task is finished the task status is received by the node manager (Step 7) and then forwarded to the scheduler (Step 8).

To perform phase-level scheduling, PRISM requires phase-level resource information for each job. In this work, we do not study the problem of job profiling as existing state-of-the-art job profilers, such as Starfish [12], can already provide accurate resource information that can be used by PRISM. While the accuracy of the profiles can affect

the performance of PRISM, we believe PRISM is ideal for environment where jobs that are executed repeatedly with the same input size, which is common in many production clusters [17], [18]. In these environments, the accuracy of the job profiles can be improved over time. In the absence of phase-level resource information (i.e. a new job that has no profile), PRISM can fall back to use task-level resource information specified for Hadoop Yarn. In this case, each phase has the same resource requirement as the task itself.

Finally, even though the flexibility of phase-based scheduling should allow the scheduler to improve both resource utilization and job performance over existing MapReduce schedulers, realizing such a potential is still a challenging problem. This is because pausing the task execution at run-time may delay the completion of the current and subsequent tasks, which may increase the job completion time (these delayed tasks are commonly referred to as stragglers [10]). Thus, the scheduler must avoid introducing stragglers when switching between phases. In the following sections, we will describe how PRISM overcomes this challenge.

IV. SCHEDULER DESIGN

This section describes the design of PRISM's phase-based scheduling algorithm. Upon receiving a heartbeat message from a node manager reporting resource availability on the node, the scheduler must select which phase should be scheduled on the node manager running on machine n, the algorithm computes the utilization of the machine using job's phase-level resource requirement it then computes a set of candidate phases the phases are schedulable on

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

the machine and selects phases in an iterative manner. In each iteration, for each schedulable phase of each job j, it computes the utility of the job.

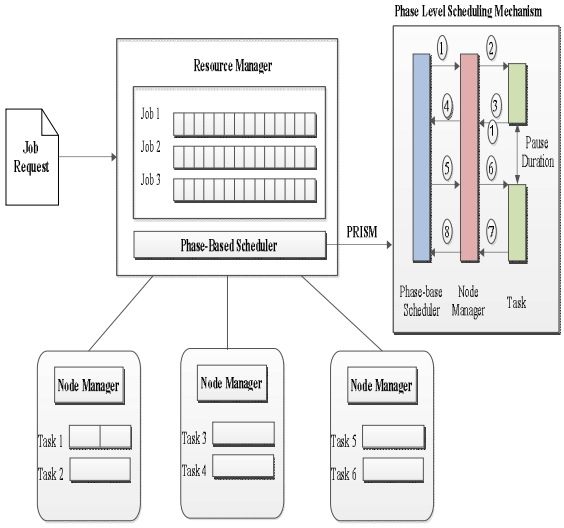

Fig 1 System architecture

A simple metric for measuring urgency is the number of seconds that a task has been paused due to phase-level scheduling. If the task has been paused for a long time, it becomes urgent to schedule its remaining phases in order to avoid creating a straggler.The utility value is phase-dependent, because phases have different dependencies. If a phase is map or shuffle, scheduling the phase implies scheduling a new map or reduce task. In this case, the utility of the phase is determined by the increase in parallelism from running an additional task. For other phases, the utility is determined by the urgency to complete the phase.

The responsibility of Map Reduce job scheduler is to assign task to machine by considering both efficiency and fairness.to achieve efficiency fob scheduler must maintain high resource utilization. Fair scheduling algorithms runs an iterative procedure by identifying users who are having highest degree of unfairness in each iteration and schedule task to users to improve the overall performance of the system. It is necessary to provide fairness without delaying the execution of each phase. To address this problem PRISM includes a solution that assigns a utility value to each phase to increase the parallelism from running an additional job. Utility value is determined by the urgency by means of seconds that a phase has been paused to avoid the stragglers.

4.1 ALGORITHM DESCRIPTION

In the scheduling algorithm node manager is responsible for reporting the resource availability on the node and it must select the next phase to should be scheduled on the node. There are J number of jobs, specifically j ∈ J and consists of two tasks called Map M and Reduce R. Resource r(t) ∈ {M,R}. The utility function is

$$U(i,n) = U\,fairness(i,n) + \alpha \,.\, Uperf(i,n)$$

The fairness of each phase is calculated as

$$U\,fairness(i,n) = U^{before}_{fairness}(i,n) - U^{after}_{fairness}(i,n).$$

$U_{fairness}$ and U_{perf} represent the utilities for improving fairness and job performance, respectively, and a is an adjustable weight factor. If we set a close to zero, then the algorithm would greedily schedule phases

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

according to the improvement in fairness. Notice that considering job performance objectives will not severely hurt fairness. When a job is severely below its fair share, scheduling any phase with non-zero resource requirement will only improve its fairness.

The following algorithm describes the scheduling algorithm used to the phase based scheduler. After receiving the status message from node manager on the machine n and the algorithm computes utilization of machine. In each schedulable phase of each lob, the algorithm computes utility function. Then select the highest utility for scheduling and for resource utilization of the machine. The algorithm repeats for all phases in the set and select the next phase to schedule. This algorithm ends when the candidate set is empty. The overall running time of the algorithm is $O(N^2\ k)$. The scalability of our algorithm can be achieved by only examining (1) the jobs that have tasks running on the machine, and (2) top k jobs with the lowest fairness measure. PRISM can naturally tolerate task failures. As phase utilities are recomputed when the scheduler tries to assign new phases to a machine, the PRISM will still make consistent decisions in spite of task failures. PRISM also supports speculative re-execution. In the implementation, even though the scheduling of a speculative task will not improve task- level parallelism, additional resources consumed will be used to improve fairness if other tasks have finished and the job is below its fair share.

Algorithm 1. Phase-Level Scheduling Algorithm

1. Upon receiving a status message from machine n:
2. Obtain the resource utilization of machine n:
3. PS ←{∅}
4. CP ←{∅}
5. Repeat
6. for each job *j*∈ jobs that hast asks on n do
7. for each schedulable phase*i*∈*j*do
8. CP ← CP ∪ {i}
9. End for
10. End for
11. For each *j*ob *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 disk i}
14. Else
15. CP ← Cp ∪ { first phase of the non-local disk 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 ← {*i*}
27. End if
28. End for
29. If *i* ← *task* with highest *U {i}* in the CP
30. *PS* ← *PS U {i}*

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

31. CP ← \ CP{*i*}
32. Update the resource utilization of machine *n*
33. End if
34. End if
35. Until CP = = ∅
36. Return *PS*

V. RELATED WORK

The original Hadoop MapReduce implements a slot-based resource allocation scheme, which does not take run-time task resource consumption into consideration. As a result, several recent works reported the inefficiency introduced due to such simple design, and proposed solutions. For instance, an adaptive resource-aware scheduler that uses job specific slots for scheduling. However, performs scheduling at task-level, and does not consider the task resource usage variations at run time. Subsequently, Hadoop Yarn represents a major endeavor towards 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.

VI. PERFORMANCE CONSIDERATIONS

PRISM has been implemented in Hadoop architecture in a cluster of 10 compute nodes. To evaluate the benefit by phase level scheduling, is necessary to compare PRISM to existing task level resource aware schedulers. In the experiments I have compared Hadoop Yarn 2.0.4 with PRISM. To compare the performance of all three schedulers (i.e., PRISM, Fair Scheduler and Yarn), we set the task container size used by Yarn according to the optimal number of slots found by Hadoop 0.20.2. The default configuration of Yarn specifies 16 virtual cores per machine. Yarn also requires that the number of vCores per task must take integer values in the job request.

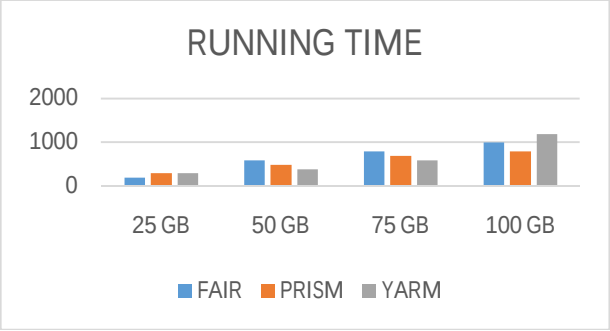

Fig 2 (a) Running a job with different input size

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

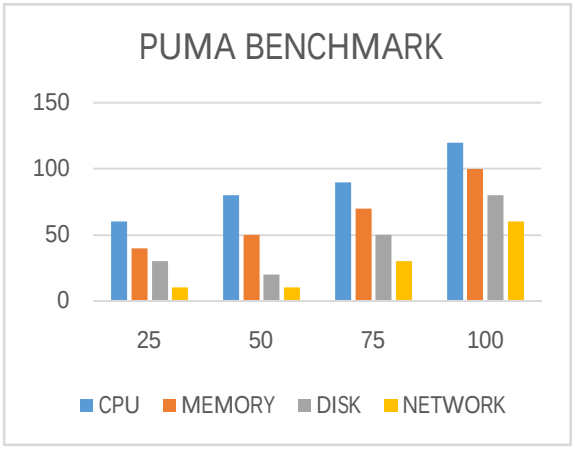

Fig 2 (b) utilization using PRISM

In these experiments, I have found the default configuration of Yarn produces lower performance compared to both the Fair Scheduler and PRISM. This is due to the large rounding errors for converting the number of vCores to integer values. Therefore, we modified the default configuration so that each machine provides 128 vCores. This significantly reduces rounding errors, allowing Yarn to produce comparable performance against both the fair scheduler and PRISM.

VII. CONCLUSION

MapReduce is a popular programming model for data intensive computing. However, despite recent efforts toward designing resource-efficient MapReduce schedulers, existing work mainly focuses on designing task-level schedulers, and 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, we introduce PRISM, a fine-grained resource-aware scheduler that coordinates task execution at the level of phases. We first demonstrate how task run-time usage can vary significantly over time for a variety of MapReduce jobs. We then present a phase-level job scheduling algorithm that improves job execution without introducing stragglers. In a 16-node Hadoop cluster running standard benchmarks, we demonstrated that PRISM offers high resource utilization and provides 1:3 improvement in job running time compared to the current Hadoop schedulers.

Lastly, we believe there are many interesting avenues for future exploration. In particular, we would like to study the problem of meeting job deadlines under phaselevel scheduling. Also, in this paper we assume all machines have identical hardware and resource capacity. 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. **"**PRISM Fine-Grained Resource-Aware Scheduling for MapReduce" Qi Zhang, Student Member, IEEE, Mohamed FatenZhani, Member, IEEE, Yuke Yang, RaoufBoutaba, Fellow, IEEE, and Bernard Wong. Appl., vol. 3, no. 2, april/june 2015
2. 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.
3. 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
4. J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.
5. 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.
6. 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.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 2, February 2016

7. Rasmussen, M. Conley, R. Kapoor, V. T. Lam, G. Porter, and A. Vahdat, "ThemisMR: An I/O-Efficient MapReduce," in Proc. ACM Symp. Cloud Comput., 2012, p. 13.
8. A. Verma, L. Cherkasova, and R. Campbell, "Resource provisioning framework for
9. MapReduce jobs with performance goals," in Proc. ACM/IFIP/USENIX Int. Conf. Middleware, 2011 , pp. 165–186.
10. D. Xie, N. Ding, Y. Hu, and R. Kompella, "The only constant is change: Incorporating time