www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issues 6 June 2016, Page No. 16880-16885

Heterogeneous Phase-Level Scheduling With Jobs Execution Scheduling Algorithm To Enhance Job Execution And Resource Management In Mapreduce

M.Sneha Priya[1], Mrs.R.Rebekha[2]

[1]M.E. Student, Department of CSE, Parisutham Institute of Technology and Science, Tamil Nadu, India
[2]Asst. Professor, Department of CSE, Parisutham Institute of Technology and Science, Tamil Nadu, India
[1]priya.sneha543@gmail.com,[2] rebeccabecky2010@gmail.com

***Abstract*— In Big Data, Map Reduce is a technique that helps to process the query from user to server in an efficient way. The Map Reduce is used to process large amount of servers in a parallel way. Hence the parallel processing is done in map reduce to retrieve results. To achieve this parallel processing, the jobs are split into 3 phases. Each phase is provided with resources for parallel and fast execution of jobs. If the resources are provided in a homogeneous way, it takes more time to complete a task.Now Heterogeneous phase level scheduling algorithm with Jobs Execution Scheduling is used to split the resources in heterogeneous way. This helps to achieve jobs to be execute greater with effective use of resources which improves speed and showing the resource usage variability within the lifetime of a task using a wide-range of Map Reduce jobs. This Scheduler improves execution parallelism and resource utilization without introducing stragglers. Energy-Efficient Algorithm provides the flow time of a job. Flow time of a job is the length of the time interval between the release time and the completion time of the job with work efficiency.**

Keywords: Bug, Bug triage, data reduction, Instance selection, Data Mining.

I. INTRODUCTION

Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. It is not a single technique or a tool, rather it involves many areas of business and technology.Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data.Corporations nowadays are more and more reliant on largescale information analytics to make important everyday enterprise selections. This shift closer to information-driven decision making has fueled the improvement of MapReduce[10], a parallel programming model that has end up synonymous with huge-scale, information-in depth computation. In MapReduce, a task is a set of Map and decrease Obligations that may be scheduled on a couple of machines, resulting in substantial discount in activity going for walks time. Many big groups,including Google, facebook, and Yahoo!, automatically use MapReduce to process huge volumes of statistics on a each day foundation. consequently, the performance and performance of MapReduce frameworks have emerge as essential to the fulfillment of nowadays's internet companies.A central component to a MapReduce system is its job scheduler. Its role is to create a schedule of Map and Reduce tasks, spanning one or more jobs, that minimizes 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.

The job scheduling trouble becomes appreciably easier to solve if we will assume that each one map tasks (and in addition, all lessen obligations) have homogenous resource necessities in terms of CPU, memory, disk and community systems, which include Hadoop MapReduce model 1.x, make this assumption to simplify the scheduling problem. these structures use a simple slot-primarily based resource allocation scheme, wherein physical resources on each system are captured by the quantity of identical slots that may be assigned to tasks. regrettably, in exercise, run-time resource intake varies from undertaking to venture and from job to activity. numerous recent research have mentioned that production workloads frequently have diverse utilization profiles and overall performance necessities[8], [21]. Failing to keep in mind those process usage characteristics can probably result in inefficient process schedules with low resource usage and lengthy jobexecution time.

Inspired by using this commentary, several latest proposals, together with useful resource-aware Adaptive Scheduling (RAS) [16] and Hadoop MapReduce version 2 (also referred to as Hadoop NextGen and Hadoop Yarn) [7], have introduced resource-conscious process schedulers to the MapReduce framework. however, those schedulers specify a hard and fast length for every assignment in phrases of required resources (e. g. CPU and reminiscence) , as a result assuming the run-time resource consumption of the mission is strong over its life time. however, this isn't actual for many MapReduce jobs. especially, it's been mentioned that the execution of every MapReduce challenge can be divided into more than one stages of data transfer, processing and storage [13]. A phase is a sub-manner in the mission that has a wonderful reason and can be characterised via the uniform useful resource intake over its duration. As we will demonstrate in section

2.2, the stages worried in the same mission will have exceptional resource call for in terms of CPU, memory, disk and network usage. consequently, scheduling obligations based totally on constant aid requirements over their durations will regularly motive either immoderate resource contention by scheduling too many simultaneous duties on a machine, or low utilization by using scheduling too few.

Big data are also highly susceptible to data complexities such as noisy, missing, inconsistent data and huge number of attributes. Data preprocessing steps such as imputing the missing values and reducing the insignificant attributes is needed for efficient classification.

CHARACTERISTICS OF BIG DATA

Volume – The quantity of data that is generated is very important in this context. It is the size of the data which determines the value and potential of the data under consideration and whether it can actually be considered as Big

Data or not. The name 'Big Data' itself contains a term which is related to size and hence the characteristic.

Variety - The next aspect of Big Data is its variety. This means that the category to which Big Data belongs to is also a very essential fact that needs to be known by the data analysts. This helps the people, who are closely analysing the data and are associated with it, to effectively use the data to their advantage and thus upholding the importance of the Big Data.

Velocity - The term 'velocity' in the context refers to the speed of generation of data or how fast the data is generated and processed to meet the demands and the challenges which lie ahead in the path of growth and development.

Variability - This is a factor which can be a problem for those who analyze the data. This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

Veracity - The quality of the data being captured can vary greatly. Accuracy of analysis depends on the veracity of the source data.

Complexity - Data management can become a very complex process, especially when large volumes of data come from multiple sources. These data need to be linked, connected and correlated in order to be able to grasp the information that is supposed to be conveyed by these data.

CONTRIBUTIONS

In this paper, I present PRISM, a phase and useful resource data-aware Scheduler for MapReduce clusters that performs aid-aware scheduling at the level of undertaking phases. in particular, we display that for maximum MapReduce programs, the runtime venture aid intake can vary substantially from segment to segment. therefore, via considering the useful resource call for on the section stage, it is possible for the scheduler to obtain better stages of parallelism while warding off resource competition. To this quit, we have developed a section-degree scheduling set of rules with the intention of reaching high process overall performance and useful resource utilization. via experiments the use of a real MapReduce cluster walking a extensive-variety of workloads, we display PRISM provides as much as 18% development in aid utilization even as permitting jobs to finish up to at least one.faster than modern-day Hadoop schedulers. in the end, even though PRISM is currently designed for Hadoop MapReduce, we accept as true with our solution may be carried out to Dryad [20] and different parallel computing frameworks as nicely.The rest of this paper is organized as follows. segment 2 offers a basic evaluation of MapReduce scheduling and task execution. We describe the phaselevel venture usage traits and our motivation in phase 3. section four introduces PRISM and describes its structure. The phase-stage scheduling set of rules is offered in details in phase 5. Our experimental evaluation of PRISM is furnished in phase 6. in the end,we summarize existing work associated with PRISM in segment 7, and draw our end in section 8.

II OVERVIEW

MapReduce [10] is a parallel computing model for huge-scale statistics-in depth computations. A MapReduce process includes kinds of obligations, specifically map and decrease duties. A map undertaking takes as input a keyvalue block saved inside the underlying dispensed report machine and runs a person-exact map characteristic to generate intermediary key-fee output. subsequently, a reduce challenge is liable for amassing and making use of a person-unique lessen feature on the collected keyvalue pairs to supply the final output.Currently, the most popular implementation of MapReduce is Apache Hadoop MapReduce [1]. A Hadoop cluster includes a huge range of commodity machines with one node serving as the master and the others performing as slaves. The grasp node runs a useful resource supervisor (additionally referred to as a activity tracker) this is responsible for scheduling obligations on slave nodes. every slave node runs a neighborhood node supervisor (also referred to as a undertaking tracker) that is accountable for launching and allocating sources for each undertaking.To do so, the task tracker launches a Java digital device (JVM) that executes the corresponding map or reduce project.

The unique Hadoop MapReduce (i.e. version 1.x and in advance) adopts a slot-based totally aid allocation scheme. The scheduler assigns tasks to every device based at the wide variety of available slots on that system. The quantity of map slots and reduce slots determines respectively the most number of map tasks and reduce obligations that may be scheduled at the gadget at a given time. As a Hadoop cluster is mostly a multi-user machine, many users can concurrently post jobs to the cluster. The activity scheduling is performed by the resource manager within the master node, which keeps a list of jobs within the gadget. each slave node monitors the development of

each running assignment and to be had sources on the node, and periodically (commonly among 1-3 seconds) transmit a heartbeat message to convey this records to the grasp node. The aid scheduler will use the furnished records to make scheduling choices. presently, Hadoop MapReduce helps numerous process schedulers which include the capability scheduler [2] and honest scheduler [3]. these schedulers make job scheduling decisions at challenge degree. They determine which assignment should be scheduled on which system at any given time, based on the range of unoccupied slots on each system. while this simple slot-based allocation scheme is simple and clean to enforce, it does no longer take runtime project resource intake into consideration.As distinct duties can also have exclusive resource necessities, this simple slot-based totally useful resource allocation scheme can result in aid competition if the scheduler assigns multiple responsibilities that have excessive demand fors unmarried resource influenced through this observation, Hadoop Yarn (additionally referred to as the Hadoop version 2 and Hadoop NextGen) [7] allows resource-aware challenge scheduling in Hadoop MapReduce clusters. at the same time as nonetheless in alpha model, it offers the potential to specify the size of the venture field (i.e. a resource reservation for a challenge

MAPREDUCE TASK LEVELS

cutting-edge Hadoop activity schedulers carry out assignment-stage scheduling, where obligations are taken into consideration because the best granularity for scheduling. but, if we examine the execution of each assignment, we will find that a project consists of more than one phases, as illustrated in figure 1. in particular, a map project may be divided into 2 major stages: map and merge2. The enter of a MapReduce task is stored as records blocks (normally of length 64MB or 128MB) in the Hadoop disbursed file device (HDFS) [4], where statistics blocks are saved across a couple of slave nodes. inside the map phase, a mapper fetches a enter information block from the Hadoop disbursed record gadget (HDFS) [4] and applies the person-defined map function on each file. The map function generates data that are serialized and collected into a buffer. while the buffer turns into full (i.e., content material size exceeds a pre-targeted threshold), the content material of the buffer might be written to the neighborhood disk. lastly, the mapper executes a merge segment to institution the output statistics based totally at the middleman keys, and save the statistics in more than one files in order that each record can be fetched a corresponding reducer.

PROBLEM STATEMENT

Minimum execution time and the homogeneous resource allocation stays a difficult problem. efficient section stage scheduling improves the resources usage and reduce the execution time.aid Allocation is based totally on phases. Map Reduce schedulers outline a static variety of slots to represent the capacity of a cluster and creates a hard and fast range of execution slots according to machine. This abstraction works for homogeneous workloads, undertaking-degree schedulers to successfully utilize available useful resource necessities of person jobs in multi-person environments.Homogeneous brief interactive queries are submitted to the same Map reduce cluster primarily based on necessities in terms of CPU, memory, disk and community bandwidth. Map reduce cluster scheduler is vital to providing the desired great of service like time efficient in excessive resource schedules.

We formally introduce our scheduling algorithm in this section. 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. Suppose there are J jobs in the system. Specifically, each job j *2 J* consists of two types of tasks: map tasks M and reduce task R. Let τ (t) *2 fM,Rg* denote the type of a task t. Given a phase i of a task t that can be scheduled on a machine n, we define the utility function of assigning a phase i to machine n as:

$$U(i, n) = Ufairness(i, n) + \alpha \_ Uperf(i, n) \quad (1)$$

where *Ufairness* and *Uperf* represent the utilities for improving fairness and job performance, respectively, and α is an adjustable weight factor. If we set α close to zero, then the algorithm would greedily schedule phases 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.Now we describe each term in Eq. (1). We define

$$Ufairness(i, n) = Ubefore\ fairness(i, n)\ Uafter\ fairness(i, n) \quad (2)$$

where *Ubefore fairness*(i, n) and *Uafter fairness*(i, n) are the fairness measures of the job before and after scheduling i on n.

Algorithm 1. Phase-Level Scheduling Algorithm

1: Upon receiving a status message from machine n:
2: Obtain the resource utilization of machine n
3: $PhaseSelected \leftarrow \{\emptyset\}$
4: $CandidateP\ hases \leftarrow \{\emptyset\}$
5: **repeat**
6: **for** each job $j \in jobsthathastasksonn$ **do**
7: **for** each scheduable phase $i \in j$ **do**
8: $CandidateP\ hases \leftarrow CandidateP\ hases \cup \{i\}$
9: **end for**
10: **end for**
11: **for** each job $j \in$ top k jobs with highest deficit n **do**
12: **if** exist schedulable data local task **then**
13: $CandidateP\ hases \leftarrow CandidateP\ hases \cup$ {first phase of the local task i}
14: **else**
15: $CandidateP\ hases \leftarrow CandidateP\ hases \cup$

*{*first phase of the non-local task *i}*
16: **end if**
17: **end for**
18: **if** *CandidateP hases≠ Ø* **then**
19: **for** *i ∈ CandidateP hases* **do**
20: **if** *i* is not schedulable on *n* given current utilization **then**
21: *CandidateP hases ← CandidateP hases\{i}*
22: continue;
23: **end if**
24: Compute the utility *U*(*i; n*) as in equation (1)
25: **if** *U*(*i; n*) ≤ 0 **then**
26: *CandidateP hases ← CandidateP hases\{i}*
27: **end if**
28: **end for**
29: **if** *CandidateP hases≠ Ø* **then**
30: *i ←* task with highest *U*(*i; n*) in the *CandidateP hases*
31:.*PhaseSelected ← PhaseSelected ∪ {i}*
CandidateP hases ← CandidateP hases\{i}
32:Update the resource utilization of machine *n*
33:**end if**
34**:end if**
35:**until** *CandidateP hases == Ø*
36:return *PhaseSelected*

III EXPERIMENTS

We have implemented PRISM in Hadoop 0.20.2. Implementing this architecture requires minimal change to the existing Hadoop architecture (around 1000 lines of code). We deployed PRISM in a compute cluster which consists of 10 compute nodes. Each compute node has 4-core 2.13GHz Intel Xeon E5606 processors, 8GB RAM, 100GB of local high speed hard drive, and runs 64-bit Ubuntu OS. The network interface card (NIC) installed on each node is capable of handling up to 1Gb/s of network traffic. Each node is connected to a top-of-rack switch and can communicate with others via a 1*Gb/s* link.We have chosen two benchmarks to evaluate the performance of PRISM:Gridmix 2 and PUMA.Gridmix 2 [5] a standard benchmark included in the Hadoop distribution. For Gridmix 2 we have chosen 3 jobs for performance evaluation: MonsterQuery (MQ), WebDataScan (WDS) and Combiner (CM). Similarly, PUMA [6] is a MapReduce benchmark developed at Purdue University. We have selected 4 jobs for performance evaluation: sort (SRT), self-join (SJ), inverted-index (II) and classification (CL). We chose these jobs because they contain a variety of resource usage characteristics. For example, sort and MonsterQuery are I/O intensive jobs, whereas Combiner and self-join are more CPU intensive. A mixture of jobs with different resource requirements allows us to better evaluate the performance of PRISM. To evaluate the benefit brought by phase-level scheduling, it is necessary to compare PRISM to existing task-level resource-aware schedulers. In our experiments, we have chosen Hadoop Yarn 2.0.4 as a competitive task-level resource-aware scheduler.Hadoop Yarn 2.0.4 is a recent version of Hadoop NextGen that allows the users to specify both CPU (i.e. number of virtual cores) and memory (i.e. GB of RAM) requirements of each task. Ideally, we would like to compare PRISM with Hadoop Yarn running a fair scheduler. However, Hadoop Yarn is yet to support fair scheduling with consideration to resource requirements. Therefore, in our experiments we compare PRISM with Hadoop Yarn Capacity scheduler, as the Capacity scheduler takes resource requirements into consideration when making scheduling decisions. Lastly, to evaluate the fairness of our scheduler, in our implementation, we adopt the same fairness metric as in the Hadoop fair scheduler 0.20.2, and use Hadoop 0.20.2 as a baseline for comparing both fairness and scheduler performance.

PERFORMANCE REQUIREMENTS

Even though job profiling is not the main focus of this work, for analysis purposes, we have implemented a simple job profiler that captures the CPU, memory and I/O usage of both tasks and compute nodes. Writing our own profiler allows us to better analyze the fine-grained resource characteristics of individual phases. In our implementation, we monitor the execution of each task and record the start and end time of every phase in the task log file. As for monitoring run-time resource usage, we rely on linux top command to record CPU and memory usage once per second. Network I/O is more difficult to profile. In our current implementation, we modified the Hadoop source code to print the values of I/O counters. The actual disk and network I/O usage over-time can be obtained from Linux utilities such as iotop and nethogs.

EVALUATION USING INDIVIDUAL JOBS

In our first experiment, our goal is to demonstrate the benefit of phase-level scheduling. For this purpose, we run a single sort job in a small cluster consisting of only 3 nodes using Fair Scheduler, Yarn and PRISM3 . The input size is set to 5*GB*. The number of map and reduce slots used by Fair Scheduler is set to 8 and 6 as discussed in previous section.The experiment results for Hadoop fair scheduler, Yarn and PRISM are shown in Figure 10, 11 and 12, respectively. In particular, the fair scheduler is able to complete the job execution in 149 seconds, whereas Yarn finishes the job in 152 seconds. In contrast, PRISM achieves the same in just 125 seconds (as shown in Figure 12(a)), resulting in a 19% reduction in job running time. To understand the reason behind the performance gain, we first plotted the CPU/Memory usage as well as disk/network I/O usage in Figure 10(b) and 10(c) for Fair Scheduler, in Figure 11(b) and 11(c) for Yarn, and in Figure 12(b) and 12(c) for PRISM. We found Yarn achieves highest utilization while performing slightly worse than the fair scheduler. The main reason is that Yarn has an additional scheduling overhead. Specifically, in order to run a new MapReduce job, scheduler need to run a job controller called Application Master [7], which will be responsible for monitoring and managing the job execution. This Application Master also consumes cluster resources at run-time, which reduce the resource capacity available for task scheduling. In contrast, PRISM delivers higher utilization for all resources. The CPU utilization of PRISM is always better than that of Fair scheduler except near the end of the execution.

EVALUATION USING BENCHMARKS

DOI: 10.18535/ijecs/v5i6.26

We now present our evaluation result using both PUMA and Gridmix 2 benchmarks. In the PUMA benchmark, we vary the number of jobs between 50 to 200 to create batch workload of different size, and run each of the batch workload 3 times using Fair scheduler, Yarn and PRISM. To provide an accurate evaluate the performance gain, in our experiments, all the jobs in the batch are simultaneously submitted to the job tracker and executed concurrently in the cluster. Theresults for job completion time is shown in Figure 15(a). It can be seen that PRISM outperforms both Fair scheduler and Yarn in
all scenarios. Furthermore, Yarn generally outperforms the Fair scheduler for large workloads, because it is more resource-aware. The locality of tasks are shown in Figure 16(a). Once again, PRISM achieves lower task locality, while deliver better performance than Fair scheduler and Yarn. Figure 17(a), 18(a) and 19(a) shows the resource utilization of the cluster during the execution of each batch for each scheduler respectively. It can be seen from the diagrams that PRISM generally provider higher resource utilization than the Fair Scheduler, and One interesting observation is that PRISM achieves higher I/O throughput than Hadoop Yarn, but slightly lower CPUutilization.

Job: sorter
Input size: 5GB, Map Count: 40, Reduce Count: 56
Map stage completion: 63s, Reduce stage completion: 147s

Phase	Map	Merge	Shuffle	Sort	Reduce
t_i (s)	7.43	1.25	9.07	0.64	9.69
CPU(%)	17.42	14.35	21.58	7.5	8.21
Mem (%)	1.35	1.40	2.11	2.37	2.33
LFS(MB/s)	3.98	34.31	5.71	11.17	5.29
HDFS(MB/s)	7.17	0	0	0	5.30
Shuffle(MB/s)	0	0	7.16	0	0

Fig. 8. An Example Job Profile: Sort Job

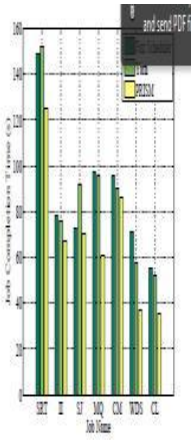
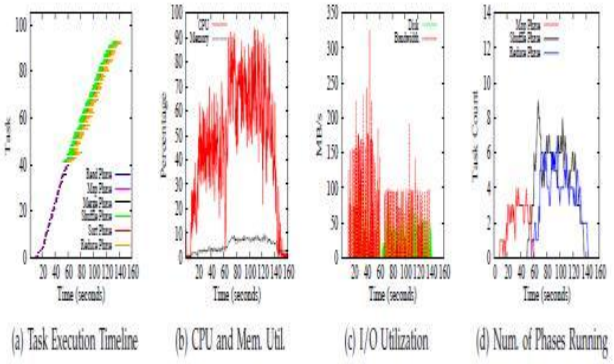
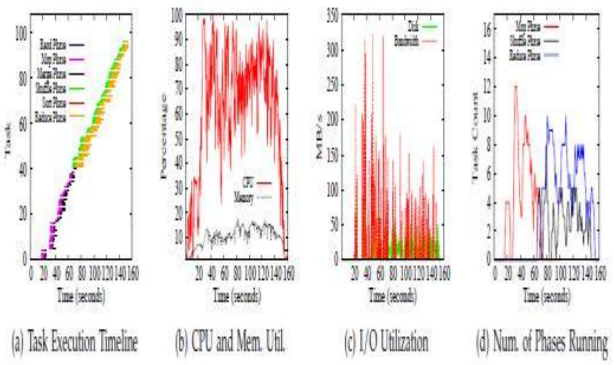


Fig. 9. Running time of each job



(a) Task Execution Timeline (b) CPU and Mem. Util. (c) I/O Utilization (d) Num. of Phases Running

Fig. 10. Sorting 5GB data with Fair-Scheduler



(a) Task Execution Timeline (b) CPU and Mem. Util. (c) I/O Utilization (d) Num. of Phases Running

V RELATED WORK

The original Hadoop MapReduce implements a slotbased 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, Polo et. al. proposed RAS [16], an adaptive resource-aware scheduler that uses job specific slots for scheduling. 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 endeavor towards resource-aware scheduling in MapReduce clusters.

It offers the ability to specify the size of each task container in terms of requiremen for each type of resources. In this context, A key 2168-7161 (c) 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
Seehttp://www.ieee.org/publications_standards/publications/right s/index.html for more information. challenge is to define the notion of fairness when multiple resource types are considered. Ghodsi et.al. proposed dominant resource fairness (DRF) as a measure of fairness in the presence of multipleresource types, and provided a simple schedulingalgorithm for achieving near-optimal DRF. However, the DRF scheduling algorithm still focuses on tasklevel scheduling, and does not consider change inresource 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 ofscheduling 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 [13]. For instance, Verma et. al. [18] 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 [13], a job profiler that collects finegrained 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. ThemsisMR [17] is another scheme that modifies MapReduce phases to improve I/O efficiency. 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 MapReduce online.

VI CONCLUSIONS AND FUTURE WORK

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 phase-level 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 heterogenous performance characteristics. Finally, improving the scalability of PRISM using distributed schedulers is also an interesting direction for future research.

ACKNOWLEDGMENTS

This work was completed as part of the Smart Applications on Virtual Infrastructure (SAVI) project funded under the National Sciences and Engineering Research Council of Canada (NSERC) Strategic Networks grant number NETGP394424-10.

REFERENCES

[1] Hadoop MapReduce distribution. http://hadoop.apache.org.

[2] Hadoop Capacity Scheduler, http://hadoop.apache.org/docs/ stable/capacity scheduler.html/.

[3] Hadoop Fair Scheduler. http://hadoop.apache.org/docs/ r0.20.2/fair scheduler.html.

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

[5] GridMix benchmark for Hadoop clusters.http://hadoop.apache.org/docs/mapreduce/current/ gridmix.html.

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

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

[8] R. Boutaba, L. Cheng, and Q. Zhang. On cloud computational models and the heterogeneity challenge. Journal of Internet Services and Applications, pages 1–10, 2012.

[9] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2010.

[10] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. Communications of the ACM, 51(1):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 USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011.

[12] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: fair allocation of multiple resource types. In USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011.

[13] 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 Conference on Innovative Data Systems Research (CIDR11), 2011.

[14] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, and K. Talwar. Quincy: fair scheduling for distributed computing clusters. In ACM SIGOPS Symposium on Operating Systems Principles (SOSP), pages 261–276, 2009.

[15] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multi-resource allocation: Flexible tradeoffs in a unifying framework. In IEEE International Conference on Computer Communications (INFOCOM), pages 1206–1214, 2012.

[16] J. Polo, C. Castillo, D. Carrera, Y. Becerra, I. Whalley, M. Steinder, J. Torres, and E. Ayguade. Resource-Aware Adaptive ´ Scheduling for MapReduce Clusters. ACM/IFIP/USENIX Middleware, pages 187–207, 2011.

[17] A. Rasmussen, M. Conley, R. Kapoor, V. T. Lam, G. Porter, and A. Vahdat. ThemisMR: An I/O-Efficient MapReduce. In ACM Symposium on Cloud Computing (SoCC), 2012.

[18] A. Verma, L. Cherkasova, and R. Campbell. Resource Provisioning Framework for MapReduce Jobs with Performance Goals. ACM/IFIP/USENIX Middleware, pages 165–186, 2011.

[19] D. Xie, N. Ding, Y. Hu, and R. Kompella. The only constant is change: incorporating time-varying network reservations in data centers. In ACM SIGCOMM, 2012.

[20] 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 USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2008.

[21] 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 European conference on Computer systems (Eurosys), 2010.

[22] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In USENIX Symposium on Operating Systems Design and Implementation (OSDI), volume 8, page 7, 2008.