

Improving MapReduce Performance in a Heterogeneous Cloud: A Measurement Study

Xu Zhao^{1,2}, Ling Liu², Qi Zhang², Xiaoshe Dong¹

¹Xi'an Jiaotong University, Shanxi, China, 710049, e-mail: zhaoxu1987@stu.xjtu.edu.cn, xsdong@mail.xjtu.edu.cn

²Georgia Institute of Technology, Atlanta, Georgia, USA, 30332, e-mail: {ling.liu, qzhang90}@cc.gatech.edu

Abstract—Hybrid clouds, geo-distributed cloud and continuous upgrades of computing, storage and networking resources in the cloud have driven datacenters evolving towards heterogeneous clusters. Unfortunately, most of MapReduce implementations are designed for homogeneous computing environments and perform poorly in heterogeneous clusters. Although a fair of research efforts have dedicated to improve MapReduce performance, there still lacks of in-depth understanding of the key factors that affect the performance of MapReduce jobs in heterogeneous clusters. In this paper, we present an extensive experimental study on two categories of factors: system configuration and task scheduling. Our measurement study shows that an in-depth understanding of these factors is critical for improving MapReduce performance in a heterogeneous environment. We conclude with five key findings: (1) Early shuffle, though effective for reducing the latency of MapReduce jobs, can impact the performance of map tasks and reduce tasks differently when running on different types of nodes. (2) Two phases in map tasks have different sensitive to input block size and the ratio of sort phase with different block size is different for different type of nodes. (3) Scheduling map or reduce tasks dynamically with node capacity and workload awareness can further enhance the job performance and improve resource consumption efficiency. (4) Although random scheduling of reduce tasks works well in homogeneous clusters, it can significantly degrade the performance in heterogeneous clusters when shuffled data size is large. (5) Phase-aware progress rate estimation and speculation strategy can provide substantial performance gain over the state of art speculation scheduler.

I. INTRODUCTION

Cloud datacenters typically employ cluster computing infrastructure for big data processing and cloud service provisioning. Hadoop MapReduce [2] clusters are one of the most popular cluster deployment in the cloud. Although compute clusters in datacenters are homogeneous by design, as CPU, memory, storage and network communication technologies advance over time, most datacenters continue to upgrade their computing infrastructure. At the same time, hybrid cloud and geo-distributed cloud become popular solution approach to instantaneous demand of additional computational resources to expand in-house resources and to maintain peak service demands. As a result, most data centers are equipped with heterogeneous sets of servers, typically ranging from fast nodes with high CPU and memory capacity to slow nodes with lower CPU and memory capacity (physical machines or virtual machines). Cloud datacenters provide different capacity of nodes at different price, represented by Amazon EC2. In order to

meet the performance requirements of different workloads at affordable cost, the mix of nodes with different compute, storage and networking capacities is needed.

Unfortunately, most MapReduce implementations are designed and optimized for homogeneous clusters and deliver unstable and poor performance on heterogeneous clusters. In recent years, a fair amount of independent research has studied some problems experienced in heterogeneous environments [4,5,7,8,10,12,13,17,18] with LATE [4] as the most popular representative. However, most existing efforts [5,17,18] only focus on speculation schedulers to find the stragglers and schedule a copy of straggler tasks on other nodes. Very little efforts have been put forward to measure the effectiveness and efficiency of setting a proper configuration for the heterogeneous environment. In addition, many research efforts on tuning MapReduce performance only analyze a subset of parameters [6,9,15]. To the best of our knowledge, there is no systematic study and in-depth analysis on the root causes of unstable and sometimes poor runtime performance of MapReduce jobs on heterogeneous clusters. For example, how much performance gains and overhead the early shuffle will have for data intensive applications with huge intermediate data on heterogeneous clusters, how different configuration parameters and their correlations may impact on MapReduce performance on heterogeneous clusters; when do existing schedulers, including Hadoop-LATE, fail to deliver good performance and why. The ability to answer these and many other related questions on performance optimization of MapReduce jobs are critical for developing the next generation cluster computing solutions.

Bearing these open issues in mind, in this paper, we present a comprehensive experimental study with in-depth analysis on two categories of performance factors: system configuration and task scheduling, especially how different configuration parameters and different scheduling parameters may interact and impact on the MapReduce performance in a heterogeneous environment. Our measurement study shows that an in-depth understanding of these factors is critical for improving MapReduce performance in a heterogeneous environment.

The rest of the paper is organized as follows. Section 2 gives a brief overview of MapReduce execution model, and the framework for distributed task scheduling. Section 3 describes the methodology of our measurement study. We present the experimental results and analysis in Section 4 to

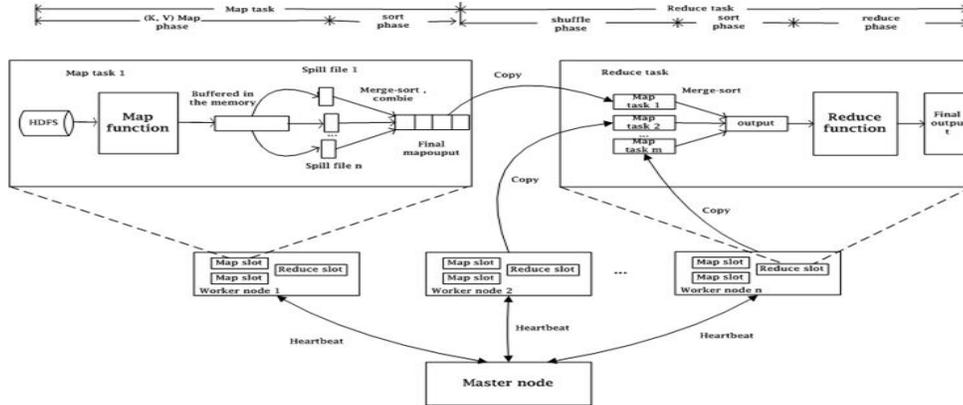


Figure 1. MapReduce execution model

experimentally analyze the hidden problems and the limitations of the current Hadoop MapReduce task scheduler (Hadoop-LATE), including the scenarios in which it performs poorly and the in-depth experimental analysis of the root causes for such limitations. We conclude the paper with a discussion on related work and a summary of our original contributions.

II. OVERVIEW AND BACKGROUND

A. MapReduce execution model

A MapReduce cluster typically consists of one master node and a set of worker nodes. The JobTracker runs on the master node for job execution and task scheduling, and the TaskTracker runs on each worker node for launching and executing tasks assigned by the JobTracker. The input data of a MapReduce job is typically divided into multiple input blocks, one per Map task. By default configuration, a worker node typically has two map slots and one reduce slot and it runs tasks on its task slots. A sketch of MapReduce operational framework is given in Figure 1. A map task consists of two phases: read and sort and its output file is stored in the local storage of the node. A reduce task consists of three phases: shuffle, sort and reduce. In the shuffle phase, a reduce task first obtain the reduce input by copying its input pieces from each map node based on its partition ID. The reduce task can only enter the sort phase after all the reduce input pieces have been copied to local storage.

B. Early shuffle v.s. no early shuffle.

No early shuffle refers to the reduce tasks cannot start shuffle phase until all map tasks of a job have completed. In contrast, early shuffle breaks this rigid bulk synchronization point by allowing the reduce task to start shuffle phase as soon as some map tasks have completed and their map output files are available. For example, the default early shuffle condition in the Hadoop MapReduce configuration is 5% of map tasks completed. Early shuffle utilizes the parallel computing to mitigate bisector network congestion

of the cluster and work well when the number of map tasks and intermediate data are large.

Although early shuffle can reduce the overall job execution time, there is no reported study to date on the additional overhead introduced by early shuffle when shuffle data is large and its impact on (i) the execution performance of concurrent map tasks, (ii) the amount of performance degradation incurred at map nodes to accommodate the early shuffle workloads, and (iii) the best scenarios where early shuffle gives the most overall performance gains.

C. Speculative execution in Hadoop

Speculative execution refers to the duplicate execution of a task that is currently running at another straggler node due to its poor performance compared to other peer nodes for the same workload (same computation task). Two key factors that may impact on the efficiency of speculative execution are the accuracy of detecting straggler nodes and the accuracy of selecting speculative execution nodes.

Hadoop Speculative scheduler is a core component of the Hadoop MapReduce middleware package. Hadoop-Late is the most recent production speculative scheduler that replaced the previous Hadoop Speculative scheduler. It incorporates the LATE strategy [4], which estimates the remaining time of each task using the longest approximate time to end. Although LATE and Hadoop LATE are simple and light-weight, the current implementation does not take into consideration of early shuffle.

III. MEASUREMENT METHODOLOGY

In this section we outline the objectives of this measurement study and the methodology we use to design, setup and conduct the experiments.

A. Objectives of the Measurement study.

The main objective of this measurement study is two folds: First, we report the performance degradations that are observed when measuring the current Hadoop Late task scheduler under varying system configurations and with

different workload characterizations. To the best of our knowledge, many of the observations have never been reported before. Second, for each of the problems observed, we design and conduct in-depth experimental analysis to identify and understand where the root causes of the problem may be, especially in the context of distributed task scheduling, the system configuration and parameter setting choices.

We argue that such an in-depth measurement analysis of the performance problems and the limitations of the Hadoop LATE scheduler can provide deep insights and optimizations for system developers to further improve the availability, the robustness and the performance of Hadoop MapReduce task scheduling. More importantly, this measurement study also provides valuable guidance on designing the next generation distributed task scheduling algorithm that can provide high overall execution efficiency for distributed long running jobs running on heterogeneous clusters with early shuffle, while minimizing unnecessary consumption of compute resources.

B. Measurement Design

In order to meet the above goals of our measurement study, we plan to focus our experiments and measurement analysis on the following two categories of issues, which have not been studied in depth in the literature but are critical to the cloud consumers to better understand how to configure their systems and clusters in the cloud data centers, and the cloud service developers to design and implement the next generation MapReduce task scheduler.

The first category is about system configuration. We argue that correct configuration is a key factor for MapReduce performance. We are interested three fundamental aspects of correct configurations: (i) Finding the problems inherent with certain default configuration parameters; (ii) Analyzing and understanding the root causes of the problems; and (iii) Learning the best practice and possibly the automated method to set the configuration parameters. We show that by conducting extensive experiments with different settings of configuration parameters, it can guide us to find the interesting correlations between different parameters and different settings of parameters, which further facilitate our analysis on the root causes of the problems found due to the design choices made in the Hadoop MapReduce middleware.

The second category is about task scheduling, a core component of Hadoop MapReduce software package. We argue that efficient task scheduling and effective system configuration are equally important for improving performance of MapReduce jobs. Although a lot of efforts have been made to improve the Hadoop LATE task scheduler, most of existing works show improvements on restricted workloads under specific configurations with much higher complexity compared to Hadoop LATE, due to the lack of in-depth understanding of the strengths and the inherent weakness in the current task scheduler. In this

measurement study, we conduct targeted experiments to identify the root causes of the inefficiency displayed by the current MapReduce task scheduler and provide in-depth analysis of the problems due to executing MapReduce jobs in a heterogeneous cloud environment from both task scheduling and system configuration perspective.

C. Experiment Environment

Our measurement results reported in this paper are conducted on a heterogeneous Hadoop0.21 cluster with three types of nodes: (i) 10 fast nodes with 64-bit Intel Quad Core Xeon E5530, 12G memory, 500G 7200 rpm Western Digital SATA disk. (ii) 3 slow nodes: Core 2 Duo, 2G memory, 250G disk. (iii) 3 slowest nodes: 64-bit Xeon 1 core 3 GHz, 2GB memory, 146G disk. And take one fast node as master node.

As we mentioned earlier, the intermediate result size such as map outputs can be a dominating factor for execution time of shuffle phase. By switching on local combine, we may reduce the size of intermediate results significantly for some applications such as WordCount. However, local combine may not be applicable to some other applications such as Hadoop TeraSort. Thus we design three types of benchmarks for our measurement study: (a) Type 1, data-intensive applications with large shuffle data: such as WordCount without local combine and InvertedIndex which takes a list of documents as input and generates word-to-document indexing. (b) Type 2, data-intensive applications with small shuffle data: e.g., WordCount with local combine turned on. (c) Type 3, compute-intensive applications with no shuffle data: e.g., Kmeans.

IV. PERFORMANCE MEASUREMENT AND ANALYSIS

In this section, we study the key factors that affect the performance of a MapReduce job running on a heterogeneous cluster. We divide the set of key factors into two categories and report our measurement results. For each result, we further provide experimental analysis and discussion to gain better understanding of the root causes. The first category of key factors is the system configuration settings. We design and conduct several sets of experiments with different combinations of configuration settings and identify strong and interesting correlations between different configuration parameters and different settings of parameters, then we conduct in-depth analysis of the root causes from the perspectives of Hadoop MapReduce implementation. The second category of key factors is related to task scheduling. Similarly, we design and conduct a suite of experiments to expose the problems of current MapReduce task scheduler and analyze the root causes from both the design of the task scheduling algorithms and the runtime execution information.

A. Performance impact of system configuration

In this section, we study the effect of three types of system configurations on the performance of MapReduce jobs. We first study the effect of early shuffle and speculation. Then we study the effect of map input block size and memory buffer size on the performance of jobs. Finally, we study the effect of varying of task slots on the performance of jobs.

1) Effect of early shuffle and speculation

To better understand the effect of early shuffle and speculation on the performance of MapReduce jobs, we consider five execution models based on whether to turn on or off early shuffle and speculative execution. To better understand the advantage of early shuffle, we use the type 1 benchmark applications with large intermediate results. We vary the block size and the input data size and set the number of reducers to 9, which equals to the number of fast

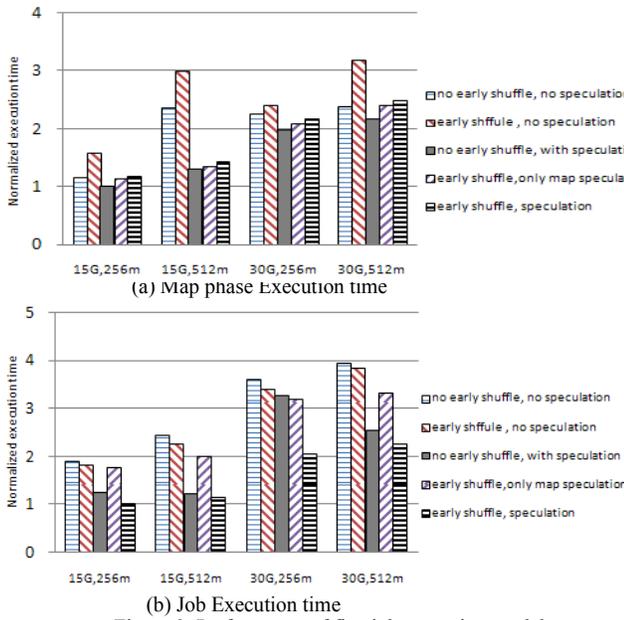


Figure 2. Performance of five job execution models

nodes. Every experimental plot is the average of 3 runs.

Figure 2 (a) shows map phase execution time of the five models with input data size varying from 15G to 30G and map input block size from 256MB to 512MB. This set of experiments shows a number of interesting observations. First, when block size increases from 256MB to 512MB, the map phase execution time is increased in all five models. This is expected. However, it is unclear whether the smaller block size will always lead to better performance. This motivates us to design another set of experiments to be discussed in the next section (see Figure 4, 5). Second, when turning on early shuffle, the map phase execution time is increased compared to no early shuffle, as expected. Also speculation can improve the effectiveness of early shuffle. The third observation is that the map phase execution time with only map speculation is more efficient than with both

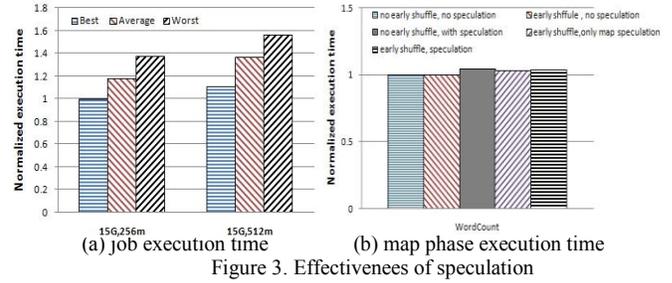


Figure 3. Effectiveness of speculation

map and reduce speculation. This implies that reduce speculation can add additional overhead on map execution.

Figure 2 (b) shows the overall job execution time in all five models. We observe that early shuffle always provides faster job response time than without early shuffle. Early shuffle combined with speculative execution offers the best performance among all five models. Also speculation can significantly improve the overall job response time, which shows the effectiveness of the current speculative scheduler, Hadoop Late. However, by looking closer at the 3rd group of histograms (30G, 256m), we make an interesting observation: the performance of the execution model with no early shuffle and speculation is surprisingly poor compared to other three groups of histograms. This indicates that the speculative execution may not offer stable performance gain (effectiveness). This motivates us to design the next set of experiments shown in Figure 3(a). By examining the best, worst and average job response time with block size of 256MB and 512MB, we observe that the speculative execution indeed shows unstable effectiveness. Also the worst case is almost 40% longer than the best case. Figure 3 (b) shows the map phase execution time for type 2 benchmark applications WordCount with local combine turned on with the input data size of 18G, block size of 256MB. We observe that speculation does not reduce the execution time, which indicates that speculation may not be always effective.

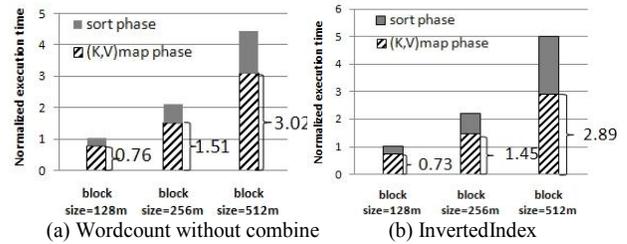


Figure 4. Map task execution time on fast node

2) Effect of block size and memory buffer size

Recall Figure 2(a), the map phase execution time will increase as the block size increases. In this section, we first study map task execution time by varying block size from 128MB, 256MB to 512MB. We use type 1 benchmark applications in the experiments reported in this section. Figure 4 shows the map task execution time of two type 1 applications on fast node, we observe that although the map

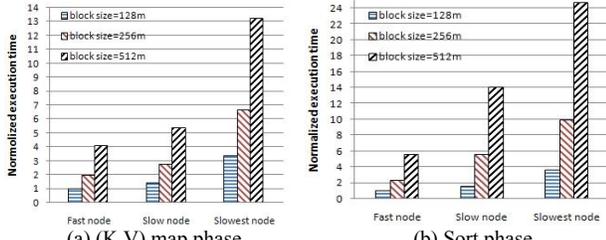


Figure 5. Two phases' ratios with different block sizes

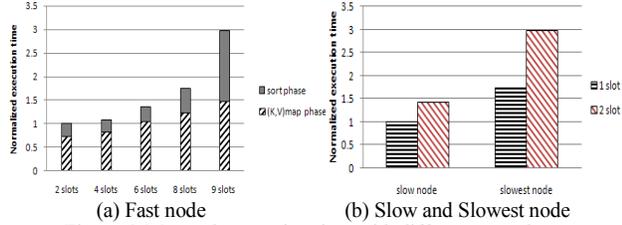


Figure 6. Map task execution time with different map slots

task execution time is not linear to input block size, the (K,V) map phase is almost linear to the input block size. Thus we design the next set of experiments to measure the execution time for each of the two phases by running on three types of nodes respectively. Figure 5 shows the results of running the application WordCount without combine. We observe that (K,V) map phase and sort phase have different sensitivity to input block size. The (K,V) map phase is linear to the input block size. However, the execution time of sort phase is not linear to input block size. Also the ratio of sort phase with different block size is different for different type of nodes

In summary, small block size can reduce the execution time of speculative map task, while bigger block size can reduce the number of map tasks. Also different types of workloads may produce different sizes of intermediate data. Although the best configuration can be different for different types of workload, it is helpful to gain an in-depth understanding of how block size, memory buffer size in map task and JVM memory size may have different impact on the execution time of map task and reduce task.

3) Effect of map slots and different type of nodes

In this section, we study the effect of different map slots for different type of nodes on the performance of MapReduce jobs. To better understand the effect of varying map slots, we run this set of experiments with no early shuffle. Figure 6 (a) shows the map task execution time by varying the number of map slots on fast nodes, ranging from 2 to 9 per map node. By increasing the allocation of map slots from 2 to 4, the map execution time has no obvious increase compared to 2 map slots. However, when the allocation of map slots is increased to 6 map slots or higher, we see that the map task execution time continue to increase quickly compared to the allocation of 2 map slots, with about 30% increase at 6 slots, 80% of increase at 8 slots and close to 200% of increase at 9 slots. One obvious

reason is that as the allocation of map slots on the fast nodes increases from 2 to 9, the number of map output files to be generated for the map tasks running on fast nodes will equally be increased. Thus it may take much longer to perform the merge-sort step for each map task. We validate this analysis by measuring CPU and memory utilization when the allocation of the map slots is changing from 2 to 9.

Figure 7 measures the CPU utilization and memory utilization on fast nodes when varying the number of map slots from 2 to 9. We see that the CPU utilization reaches 100% most of the time when the allocation of map slots is increased to 8 slots. In contrast, with 6 slots, the CPU utilization is at 90% most of the time and occasionally approaching 100%. But with 4 slots, the CPU utilization is increased to the range of 50% to 70% comparing to the CPU utilization of about 38% for 2 map slots. Interestingly, for memory utilization on fast node, we used the default JVM memory setting of 200MB when run the map task or reduce task. With the setting of 8 map slots, the memory utilization of the map task is approaching 100% only at the end of the map task execution, which is the time when merge-sort is performed. For all map slot settings, the memory utilization curves show consistently that the sort step in the map phase consumes much more memory. This set of experiments also show that CPU is the main bottleneck during the map execution when the map slots are 6 or 8 but when the allocation of map slots is increased to 9 or higher, both CPU and memory become bottleneck, though the memory utilization only reaches 100% at the end of the merge step.

Figure 8 shows the CPU and Memory utilization when varying the number of map slots on fast node for type 3 benchmark application like Kmeans. We observe that for this type of application, CPU can be the bottleneck while memory utilization is low. This set of experiments shows that assigning the map slots simply based on the capacity of memory like Yarn [3] is not always effective.

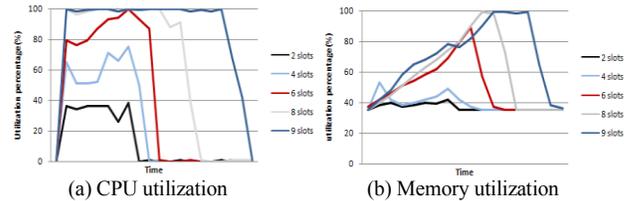


Figure 7. CPU and Memory utilization with different map task slots on fast node (WordCount without combine)

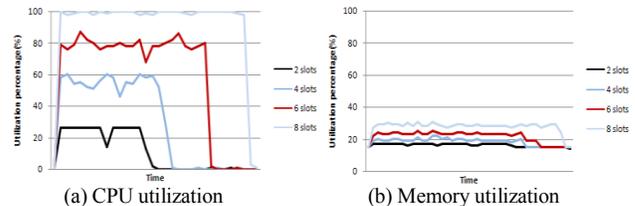


Figure 8. CPU and Memory utilization with different map task slots on fast node (Kmeans)

B. Performance impact of task scheduling algorithms

In this section, we study and analyze the problems of current task scheduler from three aspects: (i) Effect of early shuffle on map task scheduler; (ii) Effect of early shuffle on reduce task scheduler; and (iii) Effect of early shuffle on speculation task scheduler (Hadoop-LATE scheduler).

1) Effect of early shuffle on map task scheduler

When early shuffle starts, each node running map tasks may have to run the following three types of workloads concurrently:

- *Map workload*: It still runs the remaining map tasks.
- *Shuffle serving workload*: When there are map output files on this node, there will be reduce workload that serves other reduce tasks of fetching data from this node.
- *Shuffle fetching workload*: If a reduce task is running on the node, there will be reduce workload of fetching the partition of the map outputs from other map nodes.

The last two workloads may incur additional and possibly excessive burden on slow nodes and sometimes they may even slow down the progress rate of the fast node for the remaining map task. Consequently, the estimation of map progress rate may no longer be accurate, which may mislead all the decision made solely based on the progress rate, such as straggler detection and speculation task selection in the speculation scheduler.

All experiments presented in this section will measure the map task execution time with varying allocation of map slots when turn on the early shuffle. For any node that starts early shuffle, the node will run two or three type of workloads concurrently. For presentation convenience, we classify all map nodes into three types in the presence of early shuffle: 1) One workload type (Map workload); 2) Two workload type (Map workload + Shuffle serving workload); 3) Three workload type (Map workload + Shuffle serving workload + shuffle fetching workload). First we measure the map task execution time on fast node and slow node with varying number of map slots when turn on early shuffle.

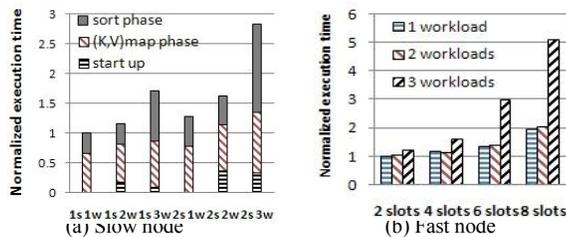


Figure 9. Map task execution time (1s,2w denotes 1 slot with 2 workloads)

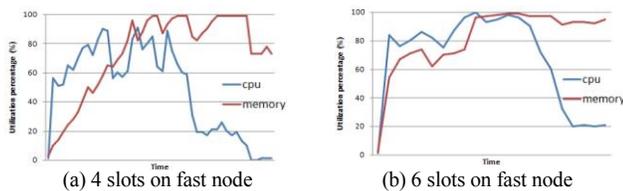


Figure 10. Varying map slots with three workloads on fast node

Figure 9 (a) shows the map execution time on slow node. We observe that no matter how many map slots are set on the slow node, when the node has 3 workloads, the map task execution time will become significantly slow. Another interesting observation is that early shuffle will cause some startup time before the (K,V) map phase actually starts. This startup time is the time spent waiting to assign the JVM. This result suggests that the scheduler should be sensitive to the types of nodes and do not assign reduce task on the slow node before it finishes its map tasks.

Figure 9 (b) shows the map execution time on fast node. As expected, as the number of map slots increases from 2 to 8, the execution time continues to slow down. We observe that when map slots is 2 and 4, the execution time of map task on the node with three workloads is only slightly slow, however, when map slots become 6 and 8, the node with three workloads become dramatically slow. We monitor the CPU and memory utilization of the node with three workloads. Figure 10 shows the allocation of 4 slots and 6 slots on fast node. We can see that when the map slot is 4, the CPU utilization does not reach 100%, and however, when map slot is 6, the CPU utilization reaches to 100%. In all cases, the memory utilization can reach up to 100%.

This set of experimental results suggests that when both CPU and memory utilization are approaching 100%, the scheduler should not schedule more map tasks on this node. Also when early shuffle is turned on, the resource utilization is dynamic and more complexity. Thus the current map scheduler that assigns all types of nodes uniformly a fixed number of map slots may fail to achieve the best performance. We need an adaptive scheduler that can fully utilize the information of node status and assign tasks dynamically.

2) Effect of early shuffle on Reduce task scheduler

In this set of experiments, we identify the problems of reduce task scheduler in the presence of early shuffle. Recall our experimental setup, we set 9 reduce tasks, which equals to the number of fast worker nodes in our heterogeneous cluster of 16 nodes. Ideally, the reduce task schedule should be able to schedule all nine reduce tasks on the nine fast nodes. However, when the reduce phase started, the reduce scheduler schedules reduce tasks randomly in the sense that the master will assign the first reduce task 0 (input data is partition 0 of all the map output files) to the first node with free reduce slot which requests for the reduce task, then assign the next reduce task 1 to the second node with free reduce slot, and so forth. Unfortunately, the input data of

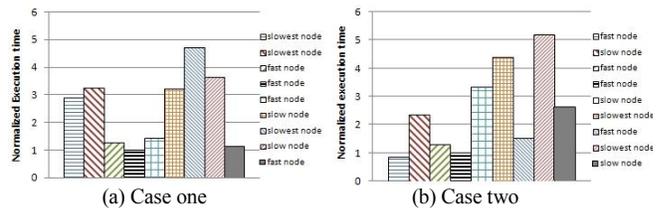


Figure 11. Two cases of reduce task execution time and reduce nodes distribution (from left to right is reduce task 0 to reduce task 8)

reduce tasks are skewed because of the partition skewedness. So the scheduler may assign the reduce task with largest input data to the slowest node, which is the worst case scenario.

Figure 11 shows the two reduce nodes distributions obtained by running the same experiment twice. The set of experiments in Figure 11 shows clearly that the assignment of reduce tasks is random. Another interesting observation is that reduce 7 with largest input data has been scheduled on a slow node in case one and a slowest node in case two. This severely hurts the performance of reduce tasks.

Although, reduce task speculation can alleviate the errors made by the reduce task scheduler to some extent, it cannot solve the problem. One approach to address this problem is that before scheduling a reduce task to the node requesting for reduce task, the reduce scheduler first checks if this node is a slow node or not. It only schedules the reduce task on it if it is not a slow node. This approach needs a new algorithm to measure the progress rate and the performance of the node.

3) Effect of early shuffle on speculation scheduler

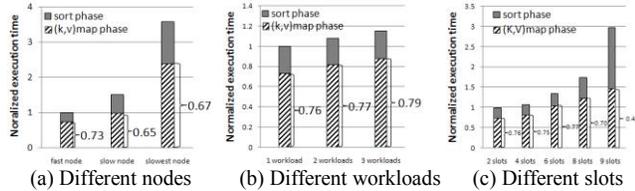


Figure 12. The ratios between (K,V) map phase and sort phase in Map task

Recall Figure 3 we have shown that the performance of speculation is not stable. The problems of current speculation scheduler can be summarized as follows:

- Waste resources: a good portion of speculative tasks cannot help reduce the execution time of the detected straggler tasks.
- Degradation of performance: In the case of eager reduce task speculation in early shuffle, the additional reduce task can hurt the map execution time on the node. In the case of wrong speculation, running the speculation task on slow node can delay the whole job execution time.

One of the main reasons for the above problems is due to the fact that Hadoop-LATE uses the fixed ratio of 2:1 between (K,V) map phase and sort phase for calculating the map progress rate, and uses the fixed ratio of 1:1:1 for the three phases (shuffle, sort and reduce) of the reduce task. Our experimental results in Figure 12 show that the ratio can be different in different type of nodes (Figure 12 (a)), different applications (Figure 4), different time in the same node (Figure 12 (b)) and different configurations (Figure 4, Figure 12 (c)). Another problem with the current speculation scheduler is its eager and simple speculation policy: When Hadoop-LATE find the task with the longest remaining time, the scheduler will assign this task to the node that has idle slot. When do early shuffle, too eager reduce speculation will affect the performance of map tasks. To provide in-

Table 1. The distribution of speculative tasks

	Slowest->Slow	Slow->Slow	Fast->Slow	Slowest->Fast	Slow->Fast	Fast->Fast
Map no early Shuffle	3	0	0	0	0	6
Map early Shuffle	0	0	2	2	4	4
Reduce early Shuffle	0	0	1	2	1	1

Table 2. Fast-> Fast case

Estimate(s)	Real(s)
217.17	138
205	102

Table 3. Slow->Fast case

Estimate(s)	Real(s)
77.58	180
51.82	153

Table 4. Slowest->Slow case

Estimate(s)	Real(s)
289.1	183

depth analysis of why current Hadoop-LATE cannot work well in the presence of early shuffle, we conduct three sets of experiments: the map speculation performance without early shuffle, the map speculation performance with early shuffle, and the reduce speculation performance. All use the type 1 benchmark application, WordCount without combine. We consider six speculation situations: Slowest \rightarrow Slow; Slow \rightarrow Slow, Fast \rightarrow Slow, Slowest \rightarrow Fast, Slow \rightarrow Fast, and Fast \rightarrow Fast. For the case Slowest \rightarrow Slow, it means that Hadoop-LATE detects the task with longest remaining time on slowest node and assign the speculative task on the slow node.

Table 1 shows the distribution of speculative tasks of the three sets of experiments. We can see that there are 9 speculative tasks when do map speculation no early shuffle, but all of them are not beneficial and have been killed. The three incorrect Fast \rightarrow Slow speculative tasks are due to the wrong slow node detection, where the scheduler does not filter the slow node from fast node, and wrong selection of slow nodes for speculative tasks.

When we turn on early shuffle and speculation, we have 12 speculative tasks (see the distribution in Table 1), 5 of which fail to improve the execution time of original map tasks, though they are correct speculative tasks. These 5 speculative tasks are distributed as 2 Fast \rightarrow Fast, 2 Slow \rightarrow Fast, 1 Slowest \rightarrow Slow. Table 2 shows the two Fast \rightarrow Fast cases, from the results we can see the actual remaining time is greater than the estimate. Table 3 shows the two Slow \rightarrow Fast cases, again the estimate remaining time is larger than the actual remaining time. Table 4 shows the case of Slowest \rightarrow Slow, the actual remaining time is still smaller. In reduce case, Fast \rightarrow Slow and Fast \rightarrow Fast denotes wrong detection of slow node and straggler when do early shuffle.

This set of experiments suggest two design principles for developing the next generation speculation scheduler: (1) we need method to calculate the progress rate of every phase to capture the dynamics of different applications, different types of nodes and different configurations. (2) We need more practical and yet accurate speculation policies that are phase aware and can filter out noises.

V. RELATED WORK

A fair amount of research efforts has been dedicated to improving the performance of MapReduce [1] in heterogeneous environments. However, most of them focus on optimizing the task scheduler or auto-tuning the configuration parameters. LATE [4] is the first to show the problems of MapReduce in heterogeneous environments and improves the native speculation scheduler by introducing the LATE method to compute the progress rate of tasks. Mantri [5], MCP [17], BASE [18] improves LATE by optimizing the speculative execution.

Hadoop auto-tuning are proposed in recent literatures [6, 9, 13, 15]. The motivation is to automatically find the optimal configuration for a job. The self-tuning system provides Job-Level tuning.

Recently, a new Hadoop version Yarn [3] is developed. In this version, the JobTracker in Hadoop 0.21 is replaced by the ResourceManager and ApplicationMaster. The ResourceManager is responsible for computing resource allocation and the application-specific ApplicationMaster is responsible for task scheduling and coordination. Our work on improving the efficiency of the task scheduling in MapReduce Job can also help system developer to improve the ApplicationMaster in Yarn.

To the best of our knowledge, none of the existing works has provided a comprehensive study of the impact of early shuffle on the effectiveness of MapReduce task scheduler and the key factors for improving the performance of MapReduce in heterogeneous environments.

VI. CONCLUSION

We have presented an in-depth measurement study on two categories of factors: system configuration and task scheduling, which are critical for improving MapReduce performance in a heterogeneous environment. We conclude with a number of key findings. First, early shuffle, though effective for reducing the latency of MapReduce jobs, can affect the performance of both map tasks and reduce tasks differently. Second, different workloads may have different sensitivity to input block size and thus adequate settings of Memory buffer size for map tasks and JVM memory size can have drastic impact on both task performance and resource utilization. Third, dynamic node capacity and workload aware scheduling map or reduce tasks can further enhance the job performance and improve resource consumption efficiency. Fourth, random scheduling of reduce tasks, though works well in homogeneous clusters, can significantly degrade the performance in heterogeneous clusters when shuffled data size is large. Finally, we conjecture that phase-aware progress rate estimation and speculation strategy can provide substantial performance gain over the state of art speculation scheduler. To the best of our knowledge, this is the first in-depth measurement and analysis on critical performance properties of MapReduce in heterogeneous environments.

ACKNOWLEDGMENT

This research is partially supported by grants from NSF CISE NetSE program, SaTC program, I/UCRC, an IBM faculty award and a grant from Intel ICST on Cloud Computing. The first author is also supported by the National Natural Science Foundation of China (No.61173039), the National High Technology Research and Development Program of China (No.2012AA01A306) and the National Natural Science Foundation for Youth Scholars of China (No.61202041).

References

- [1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Communications of the ACM*, 51 (1): 107-113, 2008.
- [2] Hadoop, <http://lucene.apache.org/hadoop>
- [3] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the Fourth ACM Symposium on Cloud Computing*. ACM, 2013.
- [4] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, Improving mapreduce performance in heterogeneous environments, in *Proc. of the 8th USENIX conference on Operating systems design and implementation*, ser. OSDI'08, 2008
- [5] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, Reining in the outliers in map-reduce clusters using mantri, in *Proc. of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10, 2010.
- [6] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, S. Babu, Starfish: A self-tuning System for Big Data Analytics, 5th Biennial Conference on Innovative Data Systems Research, CIDR11, 2011
- [7] Y. Kwon, M. Balazinska, B. Howe, J. Rolia. SkewTune: mitigating skew in mapreduce applications. In *Proc. Of the SIGMOD Conf*, May 2012
- [8] F. Ahmad, S. Chakradhar, A. Raghunathan, T. N. Vijaykumar. Tarazu: Optimizing MapReduce on Heterogeneous Clusters. ASPLOS'12, March 2012
- [9] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy. A Platform for Scalable One-Pass Analytics using MapReduce. In *Proc. of the SIGMOD Conf.*, June 2011
- [10] Z. Fadika, E. Dede, J. Hartog, M. Govindaraju. MARLA: MapReduce for Heterogeneous Clusters. CCGrid 2012.
- [11] Wenhui Lin, Jun Liu. Performance Analysis of MapReduce Program in Heterogeneous Cloud Computing. *Journal of Network*, VOL 8, No.8, August, 2013
- [12] M. Hammoud and M. Sakr. Locality-aware reduce task scheduling for mapreduce. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom 2011)*
- [13] Wenhui Lin, Jun Liu. Performance Analysis of MapReduce Program in Heterogeneous Cloud Computing. *Journal of Network*, VOL 8, No.8, August 2013
- [14] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica. Effective Straggler Mitigation: Attack of the clones. NSDI 2013.
- [15] Kun Wang, Juwei Shi, Ben Tan, Bo Yang. Automatic Task Slots Assignment in Hadoop MapReduce. ASBD 11. October 2011.
- [16] Q. Chen, C. Liu, Z. Xiao, Improving MapReduce Performance Using Smart Speculative Execution Strategy. *IEEE Transactions on Computer*, 2013.
- [17] S. Babu, "Towards automatic optimization of mapreduce programs," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 137-142
- [18] Z. H. Guo, G. Fox, M. Zhou, Y. Ruan. Improving Resource Utilization in MapReduce. 2012 IEEE International Conference on Cluster Computing (CLUSTER). pp. 402-410, 2012