# Improving Performance of Map Reduce in Heterogeneous Environments

K.Suganthi
Assistant Professor, Arasu Engineering College, India.

S.Madhavi
M.E - CSE [Final Year], Arasu Engineering College, India.

**Abstract – Map Reduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. Hadoop is an open-source implementation of Map Reduce enjoying wide adoption and is often used for short jobs where low response time is critical. Hadoop's performance is closely tied to its task scheduler implicitly assumes that cluster nodes are homogeneous and tasks make progress linearly, and uses these assumptions to decide when to speculatively re-execute tasks that appear to be stragglers. In practice, the homogeneity assumptions do not always hold. Specifically this occurs in a virtualized data center, such as Amazon's Elastic Compute Cloud (EC2).but that the Hadoop's scheduler can cause severe performance degradation in heterogeneous environments. To address this problem, a new scheduling algorithm Longest Approximate Time to End (LATE) that is highly robust to heterogeneity and it leads to improvement in response time.**

**Index Terms – Map Reduce, Virtualization, Homogeneity, Heterogeneity, Scheduler and EC2.**

## 1. INTRODUCTION

Today's most popular computer applications are Internet services with millions of users. The sheer volume of data that these services work with has led to interest in parallel processing on commodity clusters. The leading example is Google, which uses its Map Reduce framework to process 20 petabytes of data per day [1]. Other Internet services, such as e-commerce websites and social networks, also cope with enormous volumes of data. These services generate clickstream data from millions of users every day, which is a potential gold mine for understanding access patterns and increasing ad revenue. Furthermore, for each user action, a web application generates one or two orders of magnitude more data in system logs, which are the main resource that developers and operators have for diagnosing problems in production. The Map Reduce model popularized by Google is very attractive for ad-hoc parallel processing of arbitrary data. Map Reduce breaks a computation into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers. Its popular open-source implementation, Hadoop [2], was developed primarily by Yahoo, where it runs jobs that produce hundreds of terabytes of data on at least 10,000 cores [4]. Hadoop is also used at Facebook, Amazon, and Last.fm [5]. In addition, researchers at Cornell, Carnegie Mellon, University of Maryland and PARC are starting to use Hadoop for seismic simulation, natural language processing, and mining web data [5, 6]. A key benefit of Map Reduce is that it automatically handles failures, hiding the complexity of fault-tolerance from the programmer. If a node crashes, Map Reduce reruns its tasks on a different machine. Equally importantly, if a node is available but is performing poorly, a condition that we call a straggler, Map Reduce runs a speculative copy of its task (also called a "backup task") on another machine to finish the computation faster. Without this mechanism of speculative execution1, a job would be as slow as the misbehaving task. Stragglers can arise for many reasons, including faulty hardware and misconfiguration. Google has noted that speculative execution can improve job response times by 44% [1]. In this work, we address the problem of how to robustly perform speculative execution to maximize performance. Hadoop's scheduler starts speculative tasks based on a simple heuristic comparing each task's progress to the average progress. Although this heuristic works well in homogeneous environments where stragglers are obvious, we show that it can lead to severe performance degradation when its underlying assumptions are broken. We design an improved scheduling algorithm that reduces Hadoop's response time by a factor of 2. An especially compelling environment where Hadoop's scheduler is inadequate is a virtualized data center. Virtualized "utility computing" environments, such as Amazon's Elastic Compute Cloud (EC2) [3], are becoming an important tool for organizations that must process large amounts of data, because large numbers of virtual machines can be rented by the hour at lower costs than operating a data center year-round (EC2's current cost is $0.10 per CPU hour). For example, the New York Times rented 100 virtual machines for a day to convert 11 million scanned articles to PDFs [7]. Utility computing environments provide an economic advantage (paying by the hour), but they come with the caveat of having to run on virtualized resources with uncontrollable variations in performance. We also expect heterogeneous environments to become common in private

data centers, as organizations often own multiple generations of hardware, and data centers are starting to use virtualization to simplify management and consolidate servers. We observed that Hadoop's homogeneity assumptions lead to incorrect and often excessive speculative execution in heterogeneous environments, and can even degrade performance below that obtained with speculation disabled. In some experiments as many as 80% of tasks were speculatively executed.

Naively, one might expect speculative execution to be a simple matter of duplicating tasks that are sufficiently slow. In reality, it is a complex issue for several reasons. First, speculative tasks are not free they compete for certain resources, such as the network, with other running tasks. Second, choosing the node to run a speculative task on is as important as choosing the task. Third, in a heterogeneous environment, it may be difficult to distinguish between nodes that are slightly slower than the mean and stragglers. Finally, stragglers should be identified as early as possible to reduce response times. Starting from first principles, we design a simple algorithm for speculative execution that is robust to heterogeneity and highly effective in practice. We call our algorithm LATE for Longest Approximate Time to End. LATE is based on three principles: prioritizing tasks to speculate, selecting fast nodes to run on, and capping speculative tasks to prevent thrashing. We show that LATE can improve the response time of Map Reduce jobs by a factor of 2 in large clusters on EC2.

## 2. BACKGROUND

Hadoop's implementation of Map Reduce closely resembles Google's [1]. There is a single master managing a number of slaves. The input file, which resides on a distributed file system throughout the cluster, is split into even-sized chunks replicated for fault-tolerance. Hadoop divides each Map Reduce job into a set of tasks. Each chunk of input is first processed by a map task, which outputs a list of key-value pairs generated by a user defined map function. Map outputs are split into buckets based on key. When all maps have finished, reduce tasks apply a reduce function to the list of map outputs with each key. Figure 1 illustrates a Map Reduce computation. Hadoop runs several maps and reduces concurrently on each slave – two of each by default – to overlap computation and I/O. Each slave tells the master when it has empty task slots. The scheduler then assigns it tasks. The goal of speculative execution is to minimize a job's response time. Response time is most important for short jobs where a user wants an answer quickly, such as queries on log data for debugging, monitoring and business intelligence. Short jobs are a major use case for Map Reduce. For example, the average Map Reduce job at Google in September 2007 took 395 seconds [1]. Systems designed for SQL-like queries on top of Map Reduce, such as Sawzall [9] and Pig [10], underline the importance of Map Reduce for ad-hoc queries. Response time is also clearly important in a pay-by-the-hour environment like EC2. Speculative execution is less useful in long jobs, because only the last wave of tasks is affected, and it may be inappropriate for batch jobs if throughput is the only metric of interest, because speculative tasks imply wasted work. However, even in pure throughput systems, speculation may be beneficial to prevent the prolonged life of many concurrent jobs all suffering from straggler tasks. Such nearly complete jobs occupy resources on the master and disk space for map outputs on the slaves until they terminate. Nonetheless, in our work, we focus on improving response time for short jobs.

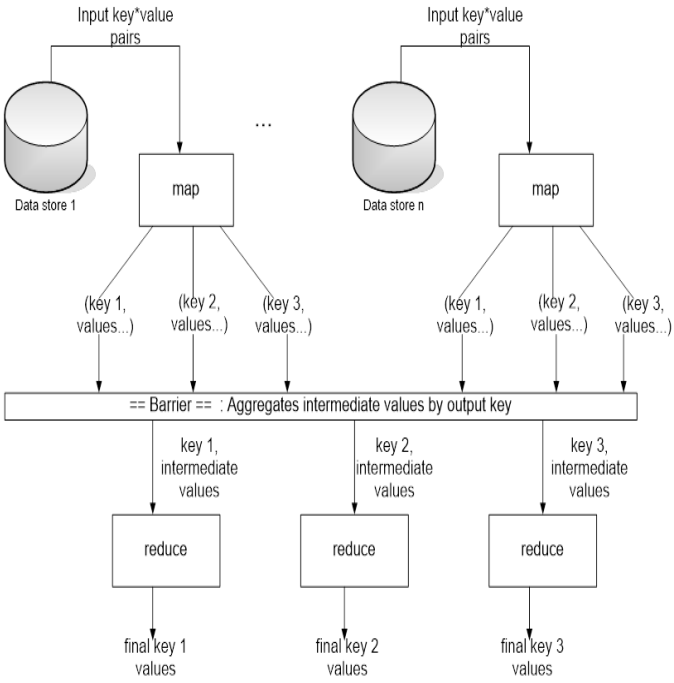


Figure 1 Map Reduce

### 2.1 Speculative Execution in Hadoop

When a node has an empty task slot, Hadoop chooses a task for it from one of three categories. First, any failed tasks are given highest priority. This is done to detect when a task fails repeatedly due to a bug and stop the job. Second, non-running tasks are considered. For maps, tasks with data local to the node are chosen first. Finally, Hadoop looks for a task to execute speculatively. To select speculative tasks, Hadoop monitors task progress using a progress score between 0 and 1. For a map, the progress score is the fraction of input data read. For a reduce task, the execution is divided into three phases, each of which accounts for 1/3 of the score:

1. The copy phase, when the task fetches map outputs.
2. The sort phase, when map outputs are sorted by key.

3. The reduce phase, when a user-defined function is applied to the list of map outputs with each key.

In each phase, the score is the fraction of data processed. Hadoop looks at the average progress score of each category of tasks (maps and reduces) to define a threshold for speculative execution: When a task's progress score is less than the average for its category minus 0.2, and the task has run for at least one minute, it is marked as a straggler. All tasks beyond the threshold are considered "equally slow," and ties between them are broken by data locality. The scheduler also ensures that at most one speculative copy of each task is running at a time. Although a metric like progress rate would make more sense than absolute progress for identifying stragglers, the threshold in Hadoop works reasonably well in homogenous environments because tasks tend to start and finish in "waves" at roughly the same times and speculation only starts when the last wave is running. Finally, when running multiple jobs, Hadoop uses a FIFO discipline where the earliest submitted job is asked for a task to run, then the second, etc. There is also a priority system for putting jobs into higher-priority queues.

2.2 Assumption in Hadoop's Scheduler

The Hadoop's scheduler makes several implicit assumptions:

1. Nodes can perform work at roughly the same rate.

2. Tasks progress at a constant rate throughout time.

3. There is no cost to launching a speculative task on a node that would otherwise have an idle slot.

4. A task's progress score is representative of fraction of its total work that it has done. Specifically, in a reduce task, the copy, sort and reduce phases each take about 1/3 of the total time.

5. Tasks tend to finish in waves, so a task with a low progress score is likely a straggler.

6. Tasks in the same category (map or reduce) require roughly the same amount of work.

As we shall see, assumptions 1 and 2 break down in a virtualized data center due to heterogeneity. Assumptions 3, 4 and 5 can break down in a homogeneous data center as well, and may cause Hadoop to perform poorly there too. In fact, Yahoo disables speculative execution on some jobs because it degrades performance, and monitors faulty machines through other means. Facebook disables speculation for reduce tasks [14]. Assumption 6 is inherent in the Map Reduce paradigm, so we do not address it in this paper. Tasks in Map Reduce should be small, otherwise a single large task will slow down the entire job. In a well-behaved Map Reduce job, the separation of input into equal chunks and the division of the key space among reducers ensures roughly equal amounts of work. If this is not the case, then launching a few extra speculative tasks is not harmful as long as obvious stragglers are also detected.

## 3. HOW THE ASSUMPTIONS BREAK DOWN

### 3.1 Heterogeneity

The first two assumptions are about homogeneity: Hadoop assumes that any detectably slow node is faulty. However, nodes can be slow for other reasons. In a non-virtualized data center, there may be multiple generations of hardware. In a virtualized data center where multiple virtual machines run on each physical host, such as Amazon EC2, co-location of VMs may cause heterogeneity. Although virtualization isolates CPU and memory performance, VMs compete for disk and network bandwidth. In EC2, co-located VMs use a host's full bandwidth when there is no contention and share bandwidth fairly when there is contention [12]. Contention can come from other users' VMs, in which case it may be transient, or from a user's own VMs if they do similar work, as in Hadoop. We measure performance differences of 2.5x caused by contention. Note that EC2's bandwidth sharing policy is not inherently harmful – it means that a physical host's I/O bandwidth can be fully utilized even when some VMs do not need it – but it causes problems in Hadoop. Heterogeneity seriously impacts Hadoop's scheduler. Because the scheduler uses a fixed threshold for selecting tasks to speculate, too many speculative tasks may be launched; taking away resources from useful tasks (assumption 3 is also untrue). Also, because the scheduler ranks candidates by locality, the wrong tasks may be chosen for speculation first. For example, if the average progress was 70% and there was a 2x slower task at 35% progress and a 10x slower task at 7% progress, then the 2x slower task might be speculated before the 10x slower task if its input data was available on an idle node. We note that EC2 also provides "large" and "extra-large" VM sizes that have lower variance in I/O performance than the default "small" VMs, possibly because they fully own a disk. However, small VMs can achieve higher I/O performance per dollar because they use all available disk bandwidth when no other VMs on the host are using it. Larger VMs also still compete for network bandwidth. Therefore, we focus on optimizing Hadoop on "small" VMs to get the best performance per dollar.

### 3.2 Other Assumptions

Assumptions 3, 4 and 5 are broken on both homogeneous and heterogeneous clusters, and can lead to a variety of failure modes. Assumption 3, that speculating tasks on idle nodes costs nothing, breaks down when resources are shared. For example, the network is a bottleneck shared resource in large Map Reduce jobs. Also, speculative tasks may compete for disk I/O in I/O-bound jobs. Finally, when multiple jobs are submitted, needless speculation reduces throughput without

improving response time by occupying nodes that could be running the next job. Assumption 4, that a task's progress score is approximately equal to its percent completion, can cause incorrect speculation of reducers. In a typical Map Reduce job, the copy phase of reduce tasks is the slowest, because it involves all-pairs communication over the network. Tasks quickly complete the other two phases once they have all map outputs. However, the copy phase counts for only 1/3 of the progress score. Thus, soon after the first few reducers in a job finish the copy phase, their progress goes from 1/3 to 1, greatly increasing the average progress. As soon as about 30% of reducers finish, the average progress is roughly $0.3 \bullet 1 + 0.7 \bullet 1/3 \approx 53\%$, and now all reducers still in the copy phase will be 20% behind the average, and an arbitrary set will be speculatively executed. Task slots will fill up, and true stragglers may never be speculated executed, while the network will be overloaded with unnecessary copying. We observed this behavior in 900-node runs on EC2, where 80% of reducers were speculated. Assumption 5, that progress score is a good proxy for progress rate because tasks begin at roughly the same time, can also be wrong. The number of reducers in a Hadoop job is typically chosen small enough so that they can all start running right away, to copy data while maps run. However, there are potentially tens of mappers per node, one for each data chunk. The mappers tend to run in waves. Even in a homogeneous environment, these waves get more spread out over time due to variance adding up, so in a long enough job, tasks from different generations will be running concurrently. In this case, Hadoop will speculatively execute new, fast tasks instead of old, slow tasks that have more total progress. Finally, the 20% progress difference threshold used by Hadoop's scheduler means that tasks with more than 80% progress can never be speculatively executed, because average progress can never exceed 100%.

## 4. THE LATE SCHEDULER

We have designed a new speculative task scheduler by starting from first principles and adding features needed to behave well in a real environment. The primary insight behind our algorithm is as follows: We always speculatively execute the task that we think will finish farthest into the future, because this task provides the greatest opportunity for a speculative copy to overtake the original and reduce the job's response time. We explain how we estimate a task's finish time based on progress score below. We call our strategy LATE, for Longest Approximate Time to End. Intuitively, this greedy policy would be optimal if nodes ran at consistent speeds and if there was no cost to launching a speculative task on an otherwise idle node. Different methods for estimating time left can be plugged into LATE. We currently use a simple heuristic that we found to work well in practice: We estimate the progress rate of each task as Progress Score/T , where T is the amount of time the task has been running for, and then estimate the time to completion as (1 – Progress Score)/Progress Rate. This assumes that tasks make progress at a roughly constant rate. There are cases where this heuristic can fail, which we describe later, but it is effective in typical Hadoop jobs. To really get the best chance of beating the original task with the speculative task, we should also only launch speculative tasks on fast nodes – not stragglers. We do this through a simple heuristic – don't launch speculative tasks on nodes that are below some threshold, Slow Node Threshold, of total work performed (sum of progress scores for all succeeded and in-progress tasks on the node). This heuristic leads to better performance than assigning a speculative task to the first available node.

Another option would be to allow more than one speculative copy of each task, but this wastes resources needlessly. Finally, to handle the fact that speculative tasks cost resources, we augment the algorithm with two heuristics:

1. A cap on the number of speculative tasks that can be running at once, which we denote Speculative Cap.

2. A Slow Task Threshold that a task's progress rate is compared with to determine whether it is "slow enough" to be speculated upon. This prevents needless speculation when only fast tasks are running.

In summary, the LATE algorithm works as follows:

1. If a node asks for a new task and there are fewer than Speculative Cap speculative tasks running:

2. Ignore the request if the node's total progress is below Slow Node Threshold.

3. Rank currently running tasks that are not currently being speculated by estimated time left.

4. Launch a copy of the highest-ranked task with progress rate below Slow Task Threshold.

Like Hadoop's scheduler, we also wait until a task has run for 1 minute before evaluating it for speculation. In practice, we have found that a good choice for the three parameters to LATE are to set the Speculative Cap to 10% of available task slots and set the Slow Node Threshold and Slow Task Threshold to the 25th percentile of node progress and task progress rates respectively. We use these values in our evaluation. We have performed a sensitivity analysis to show that a wide range of thresholds perform well.

Finally, we note that unlike Hadoop's scheduler, LATE does not take into account data locality for launching speculative map tasks, although this is a potential extension. We assume that because most maps are data-local, network utilization during the map phase is low, so it is fine to launch a speculative task on a fast node that does not have a local copy of the data. Locality statistics available in Hadoop validate this assumption.

## 5. RELATED WORK

Map Reduce was described architecturally and evaluated for end-to-end performance in [1]. However, [1] only briefly discusses speculative execution and does not explore the algorithms involved in speculative execution nor the implications of highly variable node performance. Our work provides a detailed look at the problem of speculative execution, motivated by the challenges we observed in heterogeneous environments. Much work has been done on the problem of scheduling policies for task assignment to hosts in distributed systems [18, 19].

However, this previous work deals with scheduling independent tasks among a set of servers, such as web servers answering HTTP requests.

The goal is to achieve good response time for the average task, and the challenge is that task sizes may be heterogeneous. In contrast, our work deals with improving response time for a job consisting of multiple tasks, and our challenge is that node speeds may be heterogeneous. Our work is also related to multiprocessor task scheduling with processor heterogeneity [20] and with task duplication when using dependency graphs [21].

Our work differs significantly from this literature because we focus on an environment where node speeds are unknown and vary over time, and where tasks are shared nothing. Multiprocessor task scheduling work focuses on environments where processor speeds, although heterogeneous, are known in advance, and tasks are highly interdependent due to inter task communication. This means that, in the multiprocessor setting, it is both possible and necessary to plan task assignments in advance, whereas in Map Reduce, the scheduler must react dynamically to conditions in the environment. Speculative execution in Map Reduce shares some ideas with "speculative execution" in distributed file systems [11], configuration management [22], and information gathering [23].

However, while this literature is focused on guessing along decision branches, LATE focuses on guessing which running tasks can be overtaken to reduce the response time of a distributed computation and Finally, Data Synapse, Inc. holds a patent which details speculative execution for scheduling in a distributed computing platform [15].

The patent proposes using mean speed, normalized mean, standard deviation, and fraction of waiting versus pending tasks associated with each active job to detect slow tasks. However, as discussed in earlier, detecting slow tasks eventually is not sufficient for a good response time. LATE identifies the tasks that will hurt response time the most, and does so as early as possible, rather than waiting until a mean and standard deviation can be computed with confidence.

## 6. CONCLUSION

Motivated by the real-world problem of node heterogeneity, we have analyzed the problem of speculative execution in Map Reduce. We identified flaws with both the particular threshold-based scheduling algorithm in Hadoop and with progress-rate-based algorithms in general. We designed a simple, robust scheduling algorithm, LATE, which uses estimated finish times to speculatively execute the tasks that affect the response time the most. LATE performs significantly better than Hadoop's default speculative execution algorithm in real workloads on Amazon's Elastic Compute Cloud.

## 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] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2
[4] Yahoo! Launches World's Largest Hadoop Production Application, http://tinyurl.com/2hgzv7
[5] Applications powered by Hadoop: http://wiki.apache.org/hadoop/PoweredBy
[6] Presentations by S. Schlosser and J. Lin at the 2008 Hadoop Summit. tinyurl.com/4a6lza
[7] D. Gottfrid, Self-service, Prorated Super Computing Fun, New York Times Blog, tinyurl.com/2pjh5n
[8] Figure from slide deck on MapReduce from Google academic cluster, tinyurl.com/4zl6f5. Available under Creative Commons Attribution 2.5 License.
[9] R. Pike, S. Dorward, R. Griesemer, S. Quinlan. Interpreting the Data: Parallel Analysis with Sawzall, Scientific Programming Journal, 13 (4):227-298, Oct. 2005.
[10] C. Olston, B. Reed, U. Srivastava, R. Kumar and A.Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. ACM SIGMOD 2008, June 2008.
[11] E.B. Nightingale, P.M. Chen, and J.Flinn. Speculative execution in a distributed file system. ACM Trans. Comput. Syst., 24 (4): 361-392, November 2006.
[12] Amazon EC2 Instance Types, tinyurl.com/3zjlrd
[13] B.Dragovic, K.Fraser, S.Hand, T.Harris, A.Ho, I.Pratt, A.Warfield, P.Barham, and R.Neugebauer. Xen and the art of virtualization. ACM SOSP 2003.
[14] Personal communication with the Yahoo! Hadoop team and with Joydeep Sen Sarma from Facebook.
[15] J. Bernardin, P. Lee, J. Lewis, DataSynapse, Inc. Using Execution statistics to select tasks for redundant assignment in a distributed computing platform. Patent number 7093004, filed Nov 27, 2002, issued Aug 15, 2006.
[16] G. E. Blelloch, L. Dagum, S. J. Smith, K. Thearling, M. Zagha. An evaluation of sorting as a supercomputer benchmark. NASA Technical Reports, Jan 1993.
[17] EC2 Case Studies, tinyurl.com/46vyut
[18] Mor Harchol-Balter, Task Assignment with Unknown Duration. Journal of the ACM, 49 (2): 260-288, 2002.
[19] M.Crovella, M.Harchol-Balter, and C.D. Murta. Task assignment in a distributed system: Improving performance by unbalancing load. In Measurement and Modeling of Computer Systems, pp. 268-269, 1998.
[20] B.Ucar, C.Aykanat, K.Kaya, and M.Ikinci. Task assignment in heterogeneous computing systems. J. of Parallel and Distributed Computing, 66 (1): 32-46, Jan 2006.
[21] S.Manoharan. Effect of task duplication on the assignment of dependency graphs. Parallel Comput., 27 (3): 257-268, 2001.

[22] Y. Su, M. Attariyan, J. Flinn AutoBash: improving configuration management with operating system causality analysis. ACM SOSP 2007.
[23] G. Barish. Speculative plan execution for information agents. PhD dissertation, University of Southernt California. Dec 2003

Authors

**K.Suganthi** has received B.Tech in the department of Information Technology from Anjalai Ammal Mahalingam Engineering College in the year 2005 and M.Tech in the department of Computer Science and Engineering from PRIST University in the year of 2014. And now she is working as an assistant professor in Arasu Engineering College and also have 8 year and 2 months of teaching experience in the same Institution. Her area of Interest is Mobile Computing, Network security, Wireless Network and Image Processing.

**S.Madhavi** has received B.E degree in the department of Computer Science and Engineering from As-Salam College of Engineering and Technology in the year of 2014. Now she is pursuing M.E degree in the department of Computer Science and Engineering in Arasu Engineering College. Her area of interest is Big Data, Cloud Computing, Operating System, Image Processing and Network Security.