International Journal of Grid and Distributed Computing
Vol. 9, No. 2 (2016), pp.203-214
http://dx.doi.org/10.14257/ijgdc.2016.9.2.18

A Smart Strategy for Speculative Execution Based on Hardware Resource in a Heterogeneous Distributed Environment

Qi Liu[1], Weidong Cai[1], Zhangjie Fu[2], Jian Shen[1*], Nigel Linge[3]

[1]*School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, Jiangsu, 210044, China*
[2] *Jiangsu Engineering Centre of Network Monitoring, Nanjing University of Information Science and Technology, Nanjing, Jiangsu, 210044, China*
[3] *The University of Salford, Salford, Greater Manchester, M5 4WT, UK*
qi.liu@nuist.edu.cn, n.linge@salford.ac.uk

Abstract

MapReduce, as a popular programming model for processing large data sets, has been widely applied. MapReduce 2.0 (MRV2) is a newly adopted one, which has a better performance. Those machines which have a lower performance in a cluster usually play a role who pull down the pace of job execution time. Speculative execution known as an approach for dealing with the above problems works by backing up those tasks running on a low performance machine to a higher one. Although multiple speculative execution strategies have been proposed, there are still a lot of pitfalls existing in the strategies. In this paper, Some pitfalls in proposed strategy have been modified and computer hardware has been taken into consideration (HWC-Speculation). In Hadoop-2.6, we have implemented it, called Hadoop-HWC. Experiment results show that our method can find a slow task correctly, also, the performance of MRV2 is improved.

Keywords*: MapReduce; speculative execution; computer hardware*

1. Introduction

Cloud computing has been widely studied in both academic and industry worlds due to its features of large-scale, virtualization, failure control among connected components, and synchronized communication. Requirements of a distributed system on on-demand services, computing abilities and storage resources become increasingly urgent. MapReduce [1], designed by Google, has been regarded as the most popular distribute programming model in a cloud environment.

At the same time, however, cloud systems suffer from poor load-scheduling strategies, which can consume much more execution time and temporary space than expected. While infinite computing resources can be provided in a cloud theoretically, unreasonable increment of mappers/reducers cannot achieve process efficiency, and may even waste more storage to complete. Several projects have been launched to relieve the difficulty in writing complex data analysis or data mining programs, *e.g.*, Pig [2] and Hive [3] built upon the MapReduce engines in the Hadoop environment. Optimization schemes have also been proposed [4-19].

This paper introduces a new strategy called HWC-Speculation, which can ensure the load of each reducer is relatively balanced during the reduce phrase. The total execution time is therefore reduced due to such settings. A practical Hadoop environment, called Hadoop-HWC has been implemented in our laboratory, retaining both original and refined partitioning strategies. Users can select either of them when ex[1]ecuting their actual tasks to compare the performance between them.

Jian Shen is the corresponding author.

ISSN: 2005-4262 IJGDC

According to our results, the HWC-Speculation strategy has depicted shorten execution time on both reduce phrases and entire task completion.

The rest sections of this paper are organized as followed. Related work is given in Section II. Section III describes our detailed speculative execution strategy. In Section IV, a testing environment and corresponding scenarios are designed for the verification and evaluation of the Hadoop-HWC. Finally, the conclusion and future work on our speculation execution strategies in a cloud platform are discussed in Section V.

2. Related Work

Due to the load deflection of the reduce task, it is hard to keep all tasks finishing at the same time. The lately finished task will largely pull down the speed of a job [7]. There were other research efforts conducted on partition of reducers, *e.g.* based on historical data and sampling results [10] to facilitate reduce partition processes. These methods can achieve load balancing dynamically but none of them were verified in a real Hadoop system.

Offline/online profiling was proposed to predict application resource requirements using a benchmark or real application workloads. An SVM-based prediction model, for instance, was presented in a heterogeneous environment [11]. An adaptive algorithm called HAP was designed to give out tasks based on estimated work thresholds. However, a reduce task needs to be divided into splits in the HAP, which causes extra time merging the splits and occupies more storage space in data nodes. Additionally, the training stage of the HAP consumes a large amount of time.

Deployment efficiency on virtualization has also been investigated. A general approach was introduced in [12] to estimate the resource requirements of applications running in a virtualized environment. Different types of virtualization overheads were profiled, so a regression-based model was built to map native system profiles into a virtualized system. In [13], dynamic demands of resources when starting a new VM instance were studied so that a prediction model was presented for adaptive resource provision in a cloud.

General solutions on performance evaluation and load efficiency in a cloud system have been examined and presented. PQR2, an approach to accurate performance evaluation of distributed application in a cloud was presented [14]. Jing *et al*. presented a model that can predict resource consumption of MapReduce processes based on a classification and regression tree [15].

Mars were developed to run on NVIDIA GPUs, AMD GPUs as well as multicore CPUs and implemented in Hadoop. Mars hides the programming complexity of GPUs behind the simple and familiar MapReduce interface, and automatically manages task partitioning, data distribution, and parallelization on the processors. Six representative applications also have been implemented on Mars The experimental results show that integrating Mars into Hadoop enabled GPU acceleration for a network of PCs [16]. However, the implementation is much more complex while in a cluster, the performance of GPUs and CPUs.

PrIter, a distributed computing framework was developed. The prioritized execution of iterative computations is supported in it. PrIter either stores intermediate data in memory for fast convergence or stores intermediate data in files for scaling to larger data sets. PrIter achieves up to 50 × speedup over Hadoop for a series of iterative algorithms. In addition, PrIter is shown better performance for iterative computations than other state-of-the-art distributed frameworks such as Spark and Piccolo [17]. But, the biggest pitfall is that it only adapts to iterative algorithms and cannot be applied to all algorithms.

Besides the above methods, some researchers have been studying the optimization of speculative execution strategy in MapReduce. Zaharia *et al* [18] proposed a modified version of speculative execution called Longest Approximate Time to End (LATE) algorithm that uses a different metric to schedule tasks for speculative execution. Instead of considering the progress made by a task so far, they compute the estimated time remaining, which gives a more clear assessment of a straggling tasks' impact on the overall job response time. But the time every stage occupies is not stable and the std representing standard deviation used in LATE cannot represent all cases. To solve the disadvantages in LATE, MCP [19] was proposed by QI CHEN *et al*. MCP used average progress rate to identify slow tasks while in reality the progress rate can be unstable and misleading. Straggles can be appropriately judged when there is data skew among the tasks. However, there are still a lot of pitfalls in MCP. Moreover, it can only be used in MR, not including MRV2.

All research works above have devoted themselves to particular and/or comprehensive load and usage efficiency in a distributed environment. However, in a Heterogeneous environment, reasonable scheduling is still a hard problem.

3. The HWC-Speculation Strategy

In a MRV2 cluster, after a job is submitted, a name node divides the input files into multiple map tasks, and then schedules both the map tasks and the reduce tasks to data nodes. A data node runs tasks in its containers and keeps updating the tasks' progress to the master by periodic heartbeat. Map tasks extract key-value pairs from the input, transfer them to some user-defined map function and combine function, and finally generate the intermediate map outputs. After that, the reduce tasks copy their input pieces from each map task, merge these pieces to a single ordered (key, value list) pair stream by a merge sort, transfer the stream to some user defined reduce function, and finally generate the result for the job. In general, a map task is divided into map and combine phases, while a reduce task is divided into copy, sort and reduce phases. Since Hadoop-0.20, reduce tasks can start when only some map tasks complete, which allows reduce tasks to copy map outputs earlier as they become available and hence mitigates network congestion. However, no reduce task can step into the sort phase until all map tasks complete. This is because each reduce task must finish copying outputs from all the map tasks to prepare the input for the sort phase [1].

3.1 Getting the Remaining Time of Current Task

We design a strategy stemmed from physics, the speed of MRV2 progress can be seen as the physical speed, then, thought of Newton' law easily comes into my eyesight, but it does not fit into our strategy for the speed is variable. Observing the running state, we can find that, the whole procedure can be divided in four stages: map, shuffle (copy), merge (sort) and reduce. So we calculate it through mean value of this stage with physical performance considered in and we set the weight of shuffle, merge and reduce as 1/3.

Map stage, as the first stage, the process speed can influence the speed of shuffle. The inputs of shuffle are depended on the outputs of map. As the original setting described in MRV2, shuffle stage would start when map stage has finished 5%. Then, we can get the conclusion that the speed of shuffle is variable. When shuffle starts at the very beginning, the speed may be relatively fast for that there has been a lot of inputs. As the process continues, the speed will obviously slow down, for that if the input is not bigger than the amount of container, shuffle stage will stay in

a state of waiting for data. Not until the shuffle stage has enough inputs will it continues. Concluding these, we get some formulations as follow:

$$T_{rem} = T_{cp} + T_{fp} = T_{cp} + \sum_{p}^{in} T_{avg_p} * factor_d * factor_e \quad (1)$$

$$factor_d = \frac{data_{input}}{data_{avg}}, \quad (2)$$

where T_{rem} represents the remaining time of current task, which consists of the remaining time of current phase and following phases, marked as T_{cp} and T_{fp}. All the phases contains map, shuffle, merge and reduce. T_{avg_p} represents the mean value of some phase p. p can be set 0,1, 2 and 3 which represents map, shuffle, merge and reduce phase. $data_{input}$ is the input data of that phase. $data_{avg}$ describes the mean data volume. $factor_e$ is a parameter that we will introduce it in the following part.

In our method, performance of hardware was taken into consideration. We get the parameters and the mean value of the parameters, calculate the list of ratio as show in (3), (4).

$$Ratio_i = \frac{para_i}{\overline{para_i}}, \quad (3)$$

$$List\_Ratio = \{Ratio_1, Ratio_2, ..., Ratio_i\} \quad (4)$$

Select the maximum and minimum value from the $List\_Ratio$, use A to indict the offset of the parameters of a node according to A, (5) shows the formulation of it.

$$A = (Max(List\_Ratio) - 1) * (Min(List\_Ratio) - 1), \quad (5)$$

If A is bigger than zero, then, we calculate the T_{back_p} according to (6),

$$factor_e = \sqrt[K]{\prod_{i=1}^{K} Ratio_i} * \alpha, \quad (6)$$

K is the number of the parameters and α is parameter which can be seen as a variance decided by real perfomance and in this paper, it is set 1.

When A is smaller than 0, we get another list $CList\_Ratio$, where it does not contain the maximum and minimum value of $List\_Ratio$ as shown in (7) and (8)

$$M = \{Max(List\_Ratio), Min(List\_Ratio)\} \quad (7)$$

$$CList\_Ratio = List\_Ratio - List\_Ratio \cap M, \quad (8)$$

After the step, we get the T_{back_p} according to the equation shown in (9),

$$factor_e = \frac{1}{K-2} \sum_{i=1}^{K-2} nRatio_i * \alpha. \quad (9)$$

Through the full progress of MapReduce, we can get follow formulation:

$$\int_0^T V_t dt = ProgressRate \Rightarrow \int_0^{t_1} V_{t_1} dt + \int_{t_1}^{t_2} V_{t_2} dt + ... + \int_{t_{n-1}}^{t_n} V_{t_n} dt = ProgressRate \quad (10)$$

As is described above, the speed of shuffle stage is variable, we use V_t to represent the speed at the moment t and $ProgressRate$ is the rate of progress. For that V_t is variable, the left formulation can be expressed as the right.

To further simplify the formulation as (11)

$$\sum_{i=1}^{n}\int_{t_{i-1}}^{t_i} V_{t_i}\,dt = ProgressRate, t_0 = 0 \ , \tag{11}$$

Through those equations above, we get the finishing time of shuffle phase relatively more accurate if current phase is shuffle, as shown in (12). Under this circumstance, $factor_d$ is set 1. Otherwise, total time of following phases will be largely smaller than real value, which may cause some task would not start speculative execution.

$$T_{cp} = T_{\text{shuffle}} = \frac{T_{used}\cdot V_t}{3ProgressRate\cdot \overline{V}} \tag{12}$$

If current phase is another phase, we can get T_{cp} by following equation:

$$T_{cp} = \frac{T_{used}\cdot data_{input}}{data_{copied}} \ , \tag{13}$$

where T_{back_p} represents the mean backup time of some phase and it will be detail in next section.

3.2 Method of Selecting a Backup Node and Predicting Time

The location of data can be divided into 3 classes: node-local, rack-local and no-local. As is described in Section II, performance of worker nodes is not considered accurately. To tackle this problem, Traditional strategies use the moving average process bandwidth of node-local map tasks completed on a data node to classify the node's performance. However, memory, CPU and other hardware performance are still not included while evaluating.

In our method, we take hardware into consideration while data locality is included. Before we select a backup node, we firstly get the list of data location to determine which class is the node belonged to. At the same time, we get the left memory and CPU from the Application Master, which communicates with scheduler to get the information of node list and select the node with relatively light load.

When calculating execution time of a backup task, we also accord to historical average finishing time after we have done some preprocessing work. The detailed steps are as follow:

After the step, we get the T_{back_p} according to the equation shown in (14).

$$T_{back_p} = T_{avg_p} * factor_d * factor_e \tag{14}$$

T_{back} represents the sum of backup time can be described as follow:

$$T_{back} = \begin{cases} T_{back_0} & for \quad map \quad task \\ \sum_{p=1}^{3} T_{back_p} & for \quad reduce \quad task \end{cases} \tag{15}$$

We calculate T_{avg_p} according to historical datum that other task attempts consumed. But, we directly add them in for top 10% finished tasks while for the

later, before we add new data into historical sets, we first judge if the finishing time T_{finish_p} generated by low tasks. Detailed formulation is shown as (16), if it satisfies (16), we add it to historical data and calculate mean finishing time of that phase.

$$T_{finish_p} = \begin{cases} \overline{T_{avg_p}} * (1 + ProcessRate) & For \quad map \\ \overline{T_{avg_p}} * [1 + ProcessRate - (p-1)/3] & For \quad others \end{cases} \tag{16}$$

3.3 Selection of Backup Task

We all know that not only benefits dose speculative execution have, but also costs. In a Hadoop cluster, task container can be seen as the cost of speculative execution, while the benefit is the shortening of the job execution time. In this part, we modify the cost-benefit model proposed in [12] to analyze the tradeoff. In the model, the cost is represented as the time that the computing resources are occupied, while the benefit is represented as the time saved by speculative execution. Backing up a task will occupy another container for backup time and save one container T_{rem}-T_{back}. However, it will cost just one container T_{rem} if it is not backed up. The profits of these two actions (backing it up or not) are the container occupied time that can be saved. Therefore, profits of the two actions are defined as follows:

$$profit_{backup} = \gamma * (T_{rem} - T_{back}) - 2 * \lambda * T_{back} \ , \tag{17}$$

$$profit_{backup} = T_{rem} - T_{back} \ , \tag{18}$$

where γ and λ are the weight of benefit and cost, respectively. Then we will choose the action that gains more profit. If the profit of backing up this task outweighs that of not backing it up, we consider this task slow enough and select it as a backup candidate. Otherwise we will leave the task running normally.

$$profit_{backup} > profit_{not\_backup} \Leftrightarrow \frac{T_{rem}}{T_{back}} = \frac{\gamma + 2\lambda}{\gamma + \lambda} \tag{19}$$

where $1 \le \frac{\gamma + 2\lambda}{\gamma + \lambda} \le 2$. We replace $\frac{\lambda}{\gamma}$ with ℓ to simplify the formula above. Then the formula can be expressed as $\frac{T_{rem}}{T_{back}} = \frac{1 + 2\ell}{1 + \ell}$.When a cluster is empty and there are few containers being occupied, the cost for speculative execution is less a concern, because it does not hurt other jobs' performance. On the other hand, when the cluster is busy and a lot of tasks are pending or waiting for free containers, the cost is an important consideration because backing up a task will slow down other tasks' execution. We expect that ℓ varies with the load of the cluster: $\frac{1 + 2\ell}{1 + \ell}$ gets its lowest value 1 ($\ell = 0$) when the load of the cluster is low while it reaches its highest value 2 ($\ell \to +\infty$) when the load of the cluster is high. That is to say, ℓ should increase with the load of the cluster.

Therefore, we set Ω to the $load\_factor$ of the Hadoop cluster:

$$\Omega = load\_factor = \frac{Number_{pending\_tasks}}{Number_{remaining\_containers}} \tag{20}$$

where $Number_{pending\_tasks}$ describes the number of pending tasks, and $Number_{remaining\_containers}$ represents the number of free containers can be

distributed in the cluster. At the beginning, the cluster is empty and few containers are occupied and ℓ decreases to 0, which represents there is no cost if speculative execution starts and the formula becomes $T_{rem} > T_{back}$. When the cluster is busy and a lot of tasks are pending or waiting for free containers, ℓ increases and $\frac{1+2\ell}{1+\ell}$ gradually tends to 2. Then the formula becomes $T_{rem} > 2T_{back}$. So, fewer tasks will be backed up. Therefore, using $load\_factor$ fits our demand perfectly. After all the tasks that are running but have not been backed up are iterating through, backup candidates are added into a list. The candidate that has the biggest difference between remaining time and backup time will be finally selected [19].

5. Experiment and Analysis

In order to test the performance and benefits of our load balancing strategy, we simulate a real environment. Test-bed is made up of a personal computers and a server. The server is equipped with 288 GB of memory, a 10 TB SATA driver. The personal computers is equipped with 12GB of memory, a single 500GB disk and four Intel(R) Core(TM) 2.4GHz two-core processors. On the server, we run eight virtual machines and each virtual machine is given different amounts of memory and different number of processors. The detailed information is shown in Table I.

Then, the virtual machines run as the data nodes and the server run as the name node. To evaluate the performance of load strategy, we use WordCount, Grep, Sort and Gridmix2. The top 3 are common applications used in MRV2 to evaluate strategy. Gridmix2 can be found in Apache Hadoop, which is usually used to test throughput including three small-scale jobs and three large-scale jobs. The small job is set with 10 reduce tasks and the large job is set with 70 reduce tasks. The Purdue Benchmarks Suite provides us with this application workload and we use the free datasets [20] as the workloads input. Detailed inputs of these applications are shown in Table II.

Table I. Detailed Information of each Virtual Machine

	Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8
Memory(GB)	10	8	8	8	4	8	18	12
Core processors	8	4	4	8	8	4	4	8

Table II. Input of these Applications

	WordCount	Grep	Sort
Input(GB)	50	50	30
Number of Mappers	200	132	200
Number of Reducers	16	18	15

Each of the following comparisons is the result of the jobs running six times under different strategies. To reduce the impact of the variable environment, we calculate the performance of the average, the worst, and the best cases in our results. We compare Hadoop-HWC with Hadoop- Original and Hadoop-None (speculative execution disabled) to show our performance improvement. The job execution time and the cluster throughput are seen as our primary metrics in the experiment. The improvement is calculated by the job execution time as follows:

$$Improvement = \frac{OtherStrategy - HWC}{OtherStrategy} \quad . \tag{21}$$

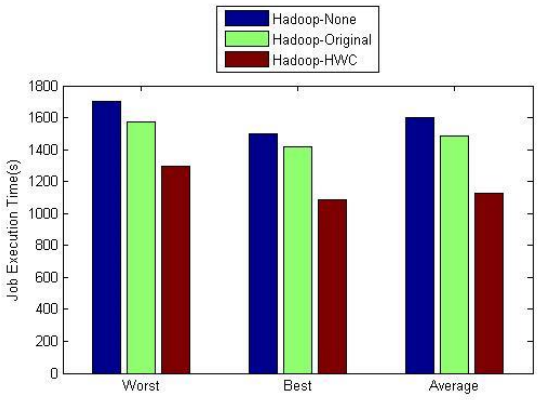

Figure 1. WrodCount

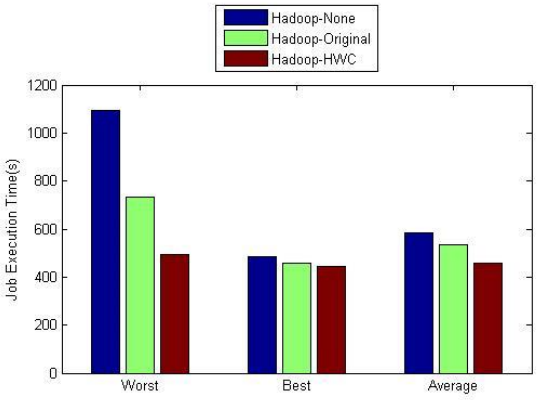

Figure 2. Grep

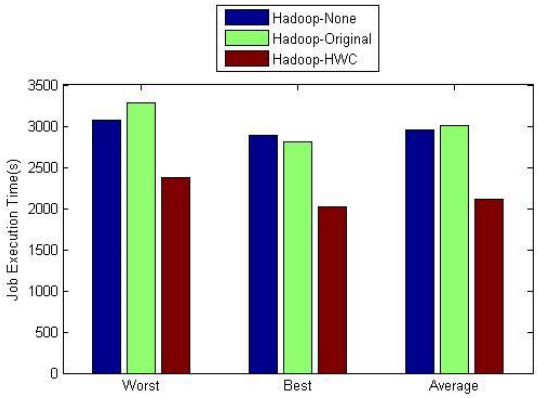

Figure 3. Sort

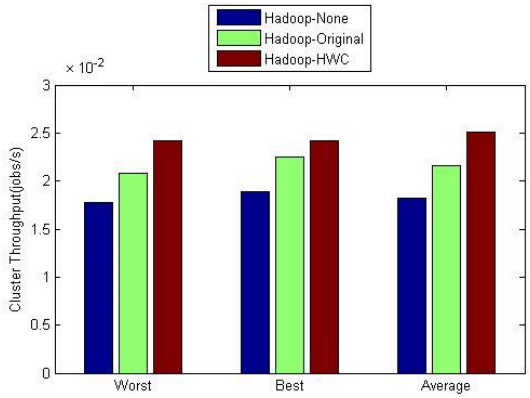

Figure 4. Gridmix2

Figure 1 shows for WordCount case, HWC finishes jobs 24% faster than Hadoop-Original and 29.4% faster than Hadoop-None on average.

Figure 2 shows that on average, HWC finishes jobs 13.7% faster than Hadoop-Original and 21.4% faster than Hadoop-None. The Grep benchmark searches a regular expression through input files and outputs the lines which contain strings matching the expression. Grep jobs are launched to search ‘table’ in the data sets from Wikipedia provided by the Purdue Benchmarks Suite. In our experiments, Grep only cost about 8 minutes before it finished the job. The short execution time may be the reason why HWC has a relatively low improvement for this case.

Figure 3 shows the job execution time of the three strategies. On average, HWC finishes jobs 29.9% faster than Hadoop-Original and 28.6%faster than Hadoop-None. To explain what leads to a higher improvement for Sort, a further analysis is given as follow. Since Sort contains much more work in reduce tasks than map tasks, the map stage makes up a very small part of the total execution time. As a result, in both map and reduce stages, HWC would have more possibilities to start effective speculative execution, under which circumstance, a higher improvement is generated.

Figure 4 shows the performance comparison of the three strategies in throughput. On average, HWC finishes jobs 20.1% faster than Hadoop-Original and 37.6% faster than Hadoop-None.

In order to further analyze the reason for our improvement, we calculate our detailed information in Table III.

Table III. Speculative Execution Comparison

Workloads	Strategy	Sum of Backup		Sum of Success		Backup success rate(%)	
		Map	Reduce	Map	Reduce	Map	Reduce
WordCount	Hadoop-Original	90	3	40	1	44.4	33.3
	Hadoop-HWC	65	12	42	10	64.6	83.3
Grep	Hadoop-Original	72	-	34	-	47.2	-
	Hadoop-HWC	35	-	30	-	85.7	-
Sort	Hadoop-Original	90	26	24	38	16	42.2
	Hadoop-HWC	87	15	51	15	57.5	100

From Table III, we can conclude that our improvement is mainly due to the precise prediction of our strategy. Compared with the original strategy in Hadoop2.6, our HWC strategy is completely better no matter for what phase and what benchmarks.

Further researches were done to find some invisible information. Figure 6, Figure 7 and Fig.8 show that HWC has an obvious superiority compared with other strategies no matter for map phase or reduce phase, which also gives an explanation for the improvement of overall job execution time.

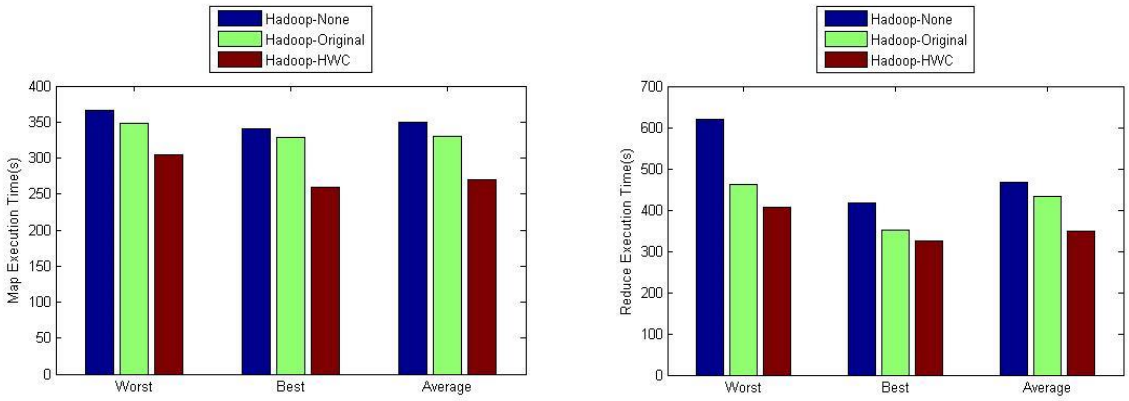

Figure 5. WordCount

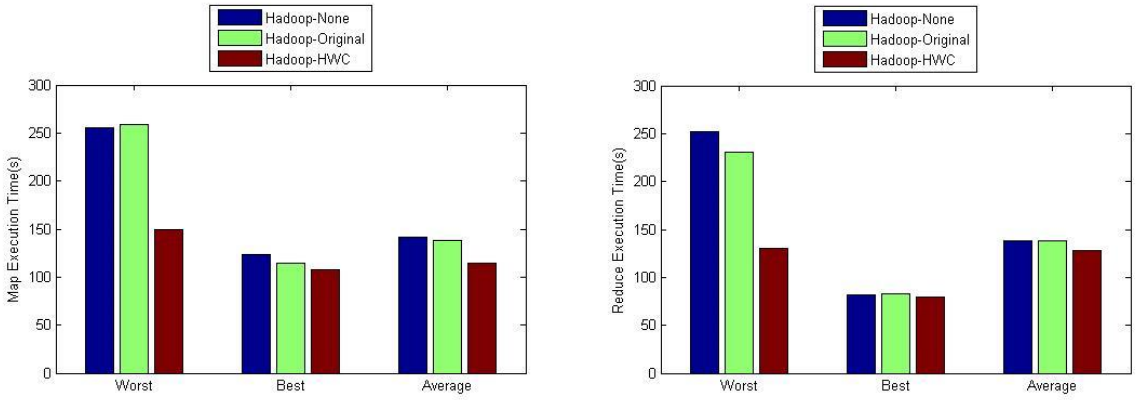

Figure 6. Grep

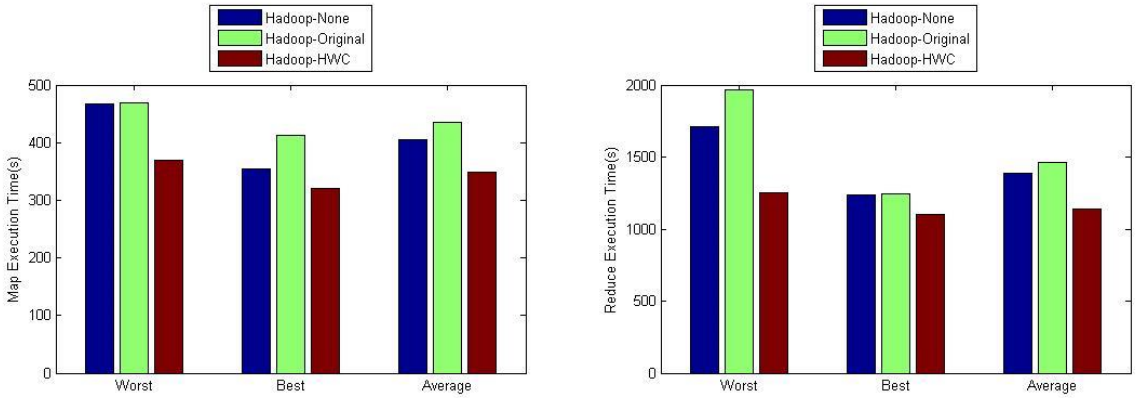

Figure 7. Sort

V. Conclusion

In this paper, we modify some pitfalls of traditional speculative strategy and take computer hardware into consideration (HWC-Speculation) and implement it in Hadoop-2.6, called it Hadoop-HWC. Experiment results show that our method can assign tasks evenly, improve the performance of MRV2 and decrease the execution time. For different benchmarks, Hadoop-HWC can get about 20% improvement compared with the original speculative execution strategy.

Acknowledgments

This work is supported by the NSFC (61300238, 61300237, 61232016, U1405254, 61373133), Basic Research Programs (Natural Science Foundation) of Jiangsu Province (BK20131004), Scientific Support Program of Jiangsu Province (BE2012473) and the PAPD fund.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski and M. Zaharia, " A view of cloud computing," Communications of the ACM, vol. 53, no. 4, **(2010)**, pp. 50-58.
[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, vol. 51, no. 1, **(2008)**, pp. 107-113.

[3] C. Olston, B. Reed, U. Srivastava, R. Kumar and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," Proceedings of the 2008 ACM SIGMOD international conference on Management of data, **(2008)**, pp. 1099-1110.
[4] R. K. Mondal, E. Nandi and D. Sarddar, "Load Balancing Scheduling with Shortest Load First, " International Journal of Grid and Distributed Computing, vol. 8, no. 4, **(2015)**, pp. 171-178.
[5] Q. Liu, W. Cai, Z. Fu, J Shen and Ni. Linge, " An Optimized Strategy for Speculative Execution in a Heterogeneous environment, " The 9th International Conference on Future Generation Communication and Networking, **(2015)**.
[6] Z. Fu, X. Sun, Q. Liu, L. Zhou and J. Shu, "Achieving Efficient Cloud Search Services: Multi-keyword Ranked Search over Encrypted Cloud Data Supporting Parallel Computing," IEICE Transactions on Communications, vol. 98, no. 1, **(2015)**,pp. 190-200.
[7] B. Palanisamy, A. Singh and L. Liu, "Cost-effective resource provisioning for mapreduce in a cloud," IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 5, **(2015)**, pp. 1265-1279.
[8] Y. Kwon, M. Balazinska, B. Howe and J. Rolia, "Skewtune: mitigating skew in mapreduce applications," Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, **(2012)**, pp. 25-36.
[9] S. Tang, B. S. Lee and B. He, "DynamicMR: A Dynamic Slot Allocation Optimization Framework for MapReduce Clusters," IEEE Transactions on Cloud Computing, vol. 2, no. 3, **(2013)**, pp. 333-347.
[10] B. Gufler, N. Augsten, A. Reiser and A. Kemper, "Handling data skew in MapReduce," Proceedings of CLOSER, **(2011)**, pp. 574-583.
[11] Y. Fan, W. Wu, Y. Xu and H. Chen, "Improving MapReduce Performance by Balancing Skewed Loads," Communications, China, vol. 11, no. 8, **(2014)**, pp. 85-108.
[12] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, **(2014)**, pp. 366-387.
[13] S. Islam, J. Keung, K. Lee and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," Future Generation Computer Systems, vol. 28, no. 1, **(2012),** pp. 155-162.
[14] A. Matsunaga and J. Fortes, " On the use of machine learning to predict the time and resources consumed by applications," Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, **(2010)**, pp. 495-504.
[15] Piao, J. Tai and J. Yan, "Computing resource prediction for mapreduce applications using decision tree," In Web Technologies and Applications, **(2012)**, pp. 570-577.
[16] W. Fang, B. He, Q. Luo and N. K. Govindaraju, "Mars: accelerating mapreduce with graphics processors," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 4, **(2010)**, pp. 608-620.
[17] Y. Zhang, Q. Gao, L. Gao and C. Wang, "Priter: a distributed framework for prioritizing iterative computations," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 9, **(2013)**, pp. 1884-1893.
[18] M. Zaharia, A. Konwinski, A. Joseph, R. Katz and I. Stoica, "Improving Mapreduce Performance in Heterogeneous Environments," OSDI, vol. 8, no. 4, **(2008)**, pp. 7.
[19] C. Qi, C. Liu and Z. Xiao, "Improving MapReduce performance using smart speculative execution strategy," IEEE Transactions on Computers, vol. 63, no. 4, **(2014)**, pp. 954-967.
[20] F. Ahmad, S. Chakradhar, A. Raghunathan and T. Vijaykumar, "Tarazu: optimizing MapReduce on heterogeneous clusters," ACM SIGARCH Computer Architecture News, **(2012)**, pp. 61-74.

Authors

Qi Liu, (M'11), He received his BSc degree in Computer Science and Technology from Zhuzhou Institute of Technology, China in 2003, and his MSc and PhD in Data Telecommunications and Networks from the University of Salford, UK in 2006 and 2010. His research interests include context awareness, data communication in MANET and WSN, smart grid and cloud computing. His recent research work focuses on intelligent agriculture, meteorological observation systems based on WSN and job scheduling in cloud computing.