J Inf Process Syst, Vol.12, No.3, pp.538~547, September 2016
http://dx.doi.org/10.3745/JIPS.03.0053

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS

A Security Protection Framework for Cloud Computing

Wenzheng Zhu* and Changhoon Lee*

Abstract
Cloud computing is a new style of computing in which dynamically scalable and reconfigurable resources are provided as a service over the internet. The MapReduce framework is currently the most dominant programming model in cloud computing. It is necessary to protect the integrity of MapReduce data processing services. Malicious workers, who can be divided into collusive workers and non-collusive workers, try to generate bad results in order to attack the cloud computing. So, figuring out how to efficiently detect the malicious workers has been very important, as existing solutions are not effective enough in defeating malicious behavior. In this paper, we propose a security protection framework to detect the malicious workers and ensure computation integrity in the map phase of MapReduce. Our simulation results show that our proposed security protection framework can efficiently detect both collusive and non-collusive workers and guarantee high computation accuracy.

Keywords
Cloud Computing, Collusive Worker, Malicious Worker, MapReduce, Non-collusive Worker

1. Introduction

Cloud computing is a recent trending in IT that moves computing and data away from desktop and portable PCs into large data centers [1]. It refers to applications delivered as services over the internet as well as to the actual cloud infrastructure. The hardware and system software in data centers provide these services. Google first developed MapReduce as a parallel computing framework to perform distributed computing on a large number of commodity computers [2]. Since then, it has gained popularity as a cloud computing framework on which to perform automatically scalable distributed applications.

Security is an area of cloud computing that presents some special challenges. Malicious workers, who can generate incorrect results in order to sabotage the output without being detected, can exploit some vulnerability. If several malicious workers are assigned to execute the same task, they can discover each other and can cheat. We call these malicious workers “collusive workers.” If the malicious workers cannot discover the others and cheat, we call them “non-collusive workers.” Hence, ensuring the integrity of MapReduce computation in cloud environments is of great importance.

There is a lot of research on cloud computing security. But the great majority of this is on data storage security, data transmission security, application security, and security-related third-party resources.

Manuscript received May 26, 2015; first revision October 7, 2015; accepted October 15, 2015.
Corresponding Author: Wenzheng Zhu (zhu_wenzheng@hotmail.com)
* Dept. of Computer Science and Engineering, Konkuk University, Seoul, Korea (zhu_wenzheng@hotmail.com, chlee@konkuk.ac.kr)

Relatively few works have been presented on the topic of malicious workers in cloud computing. In this paper, we present a security protection framework to provide integrity in the map phase of MapReduce. This framework can detect the malicious workers in order to protect cloud computing security.

This paper is organized as follows: in Section 2, we describe the MapReduce model and malicious workers. In Section 3, we present our proposed framework. In Section 4, we present our simulation experiment results to illustrate the function achieved by the proposed framework. Finally, in Section 5, we present our conclusion.

2. Related Work

The internet has changed computing to cloud computing. MapReduce is a great programming model in cloud computing that was introduced by Google. The traditional architecture of MapReduce consists of one master and a large number of workers. But the workers are vulnerable to being attacked by malicious workers, who can tamper with computation integrity. In this section, we present a review of MapReduce and malicious workers.

2.1 MapReduce Model

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real world tasks [3]. Users specify their computation in terms of a map and a reduce function. The underlying runtime system automatically parallelizes the computation across large scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks.

Users submit jobs to a scheduling system. A MapReduce job usually splits the input dataset into independent chunks, which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and output of the job are stored in a Distributed File System (DFS) [4]. The framework takes care of scheduling tasks and monitoring them as well as re-executing the failed tasks. Fig. 1 shows the overall workflow of a MapReduce operation.

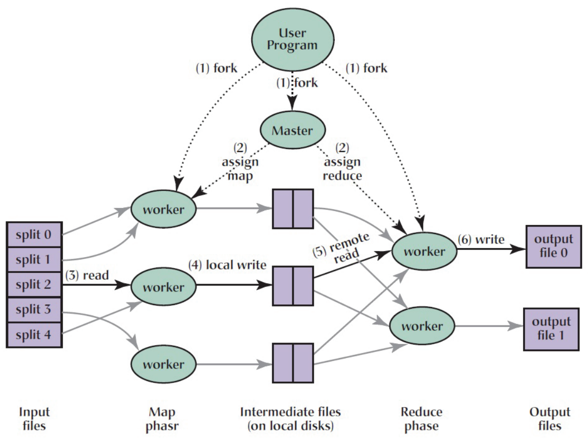


Fig. 1. MapReduce execution overview.

The master is responsible for controlling the computation, such as job management, task scheduling, and load balance. Workers are hosts that contribute computation resources to execute tasks assigned by the master. There are two phases (map phase and reduce phase) in MapReduce computation process, and in the map phase, the workers are called mappers. Similarly, in the reduce phase, the workers are called reducers [5].

First, the user starts the MapReduce job for the master programmer. In the map phase, the input data is split into pieces which are normally 16–64 MB in size, and is assigned by the master to map tasks associated with different mappers for paralleled processing [6]. The map tasks perform user-specified computations on each input key/value pair from the partition of the input data assigned to the task and generate a set of intermediate results for each key. The intermediate result for each mapper is stored in its local file system and will be stored by the keys with intermediate results from other mappers, and it divides this result data into regions to be processed by the reduce task. In the reduce phase, the intermediate results are distributed to the reducers in order to reduce the tasks associated with reducers. Each reduce task specifies which region a reducer should process [7]. After a reducer receives a reduce task, the reducer waits for notifications about map task completion events from the master and then receives intermediate results as its input. When the reducer reads its region from each mapper who finishes their map task, the reducer starts to process them, and then each reducer outputs its result to the DFS. Finally, the reducer notifies the master that returns the result to the user job.

2.2 Malicious Worker

We consider the malicious workers to be the attackers. They can cheat on a task by giving a wrong result or tamper with the intermediate result to mess up the final result. Malicious workers can be classified into two types [8], as listed below.

Non-collusive workers: This kind of worker misbehaves without coordinating with other malicious workers and returns erroneous results in order to sabotage the final job results. If one task is assigned to several workers, they return random and incorrect results.

Collusive workers: This kind of worker has more complex malicious behavior. They can communicate to behave cooperatively and make an agreement between two or more collusive workers. For the same task, they can generate and return the incorrect results with the same value to increase the probability that the master accepts their results.

To detect malicious workers in cloud computing, we created a new security protection framework. This framework consists of a replicate task and verification approach. Whenever the task queue is not empty, the master will pick one task and send it to any two workers. That is to say, two random workers execute this task, which is called a replicate task, respectively. After getting the two results from the two workers, the master can compare them. If the results are different, it means that at least one worker is a non-collusive worker. However, it is hard to tell which one is the non-collusive worker, but we can detect and pick the non-collusive worker later. The replicate task is useless to collusive workers. For detecting collusive workers, we improved the replicate task so that it is a credit-based replicate task, and we added a verifier (trusted worker) to verify the intermediate results. We will introduce this framework in detail in the next section.

3. Proposed Security Protection Framework

In this section, we present the security protection framework to detect the malicious workers in order to provide a security environment for cloud computing. Our proposed security protection framework consists of two mechanisms: credit-based replicate tasks and verification, which can combine with each other to be an efficient framework. In this section, we describe this framework in detail.

3.1 Framework Structure

In this framework, some of the components included are the master, workers (mappers and reducers), verifier, history cache, and result buffer, as shown in Fig. 2.

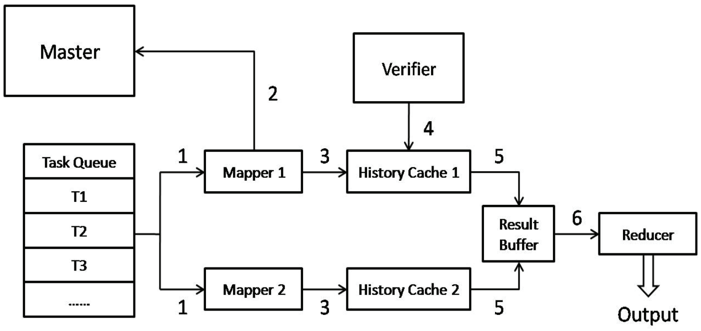


Fig. 2. Proposed framework structure.

Mapper 1 and mapper 2 are any two workers who perform the replicate tasks in the map phase. The master picks one task from the task queue and assigns it to mapper 1 and mapper 2 (Step 1). After execution, the two mappers will return their results to the master. The master compares the results to detect if they are consistent. When the results are different, it means that at least one mapper is the non-collusive worker. But it is hard to tell which one is the non-collusive worker. Then, the master will divide the two mappers into blacklist 1 and reschedule the replicate task to any two new mappers (Step 2). If the results are consistent, the results will be stored in their history caches, respectively (Step 3). The master asks the verifier (trusted worker) to verify this consistent result with a certain probability because of the verifier's limited resources. If the verification fails, the verifier returns a different result, and the master can confirm that the two mappers are collusive workers and divides them into blacklist 2. Simultaneously, all of the results stored in their history cashes will be cleared, and the tasks they have been assigned will be rescheduled (Step 4). If the verification is a success, it means that the two mappers have passed one quiz, which means that they can accumulate one credit. Once a mapper has accumulated enough credits, its history cache will release all the intermediate results to the result buffer. Then, the credit of this mapper will be reset to 0 (Step 5). Finally, when the result buffer receives two results from all the replicate tasks, it will release the results to the reducer (Step 6).

After the operation above is completed, we can use the verifier to discern and pick the non-collusive workers in blacklist 1. The mappers in blacklist 1 will execute a task with the verifier. When the result returned by a mapper is similar to the result returned by the verifier, this mapper can be released from blacklist 1. Otherwise, the mapper can be confirmed as a non-collusive worker, and it will be divided into blacklist 3. As such, we can detect the malicious workers and discriminate between the non-collusive worker and the collusive worker, where the workers in blacklist 2 are collusive workers and the workers in blacklist 3 are non-collusive workers.

3.2 Framework Algorithm

The non-collusive workers can be detected immediately by result comparison. Therefore, we only need to consider how to detect the collusive workers. In the following analysis, we focused on the environment consisting of collusive workers.

We define *m* as the malicious ratio out of all workers, *c* as the collusive worker ratio out of malicious workers, p_1 as the probability that two collusive workers execute the same replicate task (they are able to collude), p_2 as the probability that the two collusive workers are determined to commit a cheat, and p_3 as the probability that a non-collusive mapper is determined to commit a cheat. For a collusive worker, one of the three scenarios listed below must exist when it wants to pass one quiz.

Scenario a: The collusive worker executes the replicate task with another collusive worker (with probability $m*c$), and they are determined not to commit a cheat (with probability $1-p_2$). In this case, the probability is:

$$A = m*c*p_1*(1-p_2) \tag{1}$$

Scenario b: The collusive worker executes the replicate task with a non-collusive worker (with probability $m*(1-c)$), and this non-collusive worker is determined not to commit a cheat (with probability $1-p_3$). In this case, the probability is:

$$B = m*(1-c)*(1-p_3) \tag{2}$$

Scenario c: The collusive worker executes the replicate task with a healthy worker. In this case, the probability is:

$$C = 1-m \tag{3}$$

We defined the quiz threshold as *k*. If a collusive worker wants to gain the master's trust, it must pass *k* quizzes. For any case in which scenarios a, b, and c, happen *i*, *j*, and *k-i-j* times, the probability of each combination in this case is $A^i B^j C^{k-i-j}$. For this case, there are $\binom{k}{i}\binom{k-i}{j}$ combinations, so the probability of this case is:

$$P(i,j,k) = \binom{k}{i}\binom{k-i}{j} A^i B^j C^{k-i-j} \tag{4}$$

If a replicate task is assigned to two collusive workers, and either of them fails to pass *k* quizzes, the master will not release their results to the reducer. If a replicate task is assigned to a collusive worker and a non-collusive worker and the non-collusive worker commits a cheat, the master will not release their results to the reducer too. The master will release an incorrect result to the reducer when the two collusive map workers pass *k* quizzes and they cheat in a collusive manner. We defined *Δ* as the probability that a collusive worker will pass all of the *k* quizzes. With *Δ*, we can derive the cheat probability *CP*.

$$CP = m^2 * c^2 * p_1 * p_2 * \Delta^2 \tag{5}$$

As such, we can get the accuracy probability *AP* that a task will return a correct result to the master and that the master will release it to the reducer.

$$AP = 1 - CP \tag{6}$$

Also, the *AP* can be described as:

$$AP = 1 - m^2 * c^2 * p_1 * p_2 * \Delta^2 \tag{7}$$

We set *m* to 0.1, 0.3, and 0.5, respectively, and set the verification probability of the verifier to 0.25. Fig. 3 shows the relationship between the quiz threshold and accuracy based on Formula (7). In this case, *c* is 0.5, p_1 is 0.5, p_2 is 0.5, and p_3 is 0.5. From Fig. 3, we can observe that the accuracy increases when the quiz threshold increases from 1 to 7. When the quiz threshold is 7, the accuracy can reach 100%.

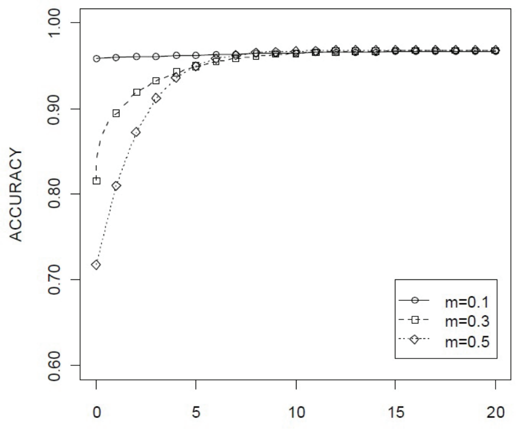


Fig. 3. Relationship between accuracy and quiz threshold.

We were able to get the relationship between accuracy and the quiz threshold based on a framework algorithm. We show our verification of this result based on a simulation experiment in the next section.

4. Simulation Experiment and Results

Our simulation experiment was obtained using Hadoop, which is an open-source software framework written in Java for the distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. The testing environment consisted of 10 computers. The computer configuration was Intel core duo 2.7G CPU, with 2G DDR3 memory, and Linux operating system. One of the computers was assigned to be the master, one as the verifier, and the others as workers.

Because the non-collusive workers can be easily detected from blacklist 1 by the verifier, we focused on detecting collusive workers in our simulation experiment.

We obtained our experimental results, as shown in Table 1, when we set the malicious ratio to 0.1.

Table 1. Malicious ratio is set to 0.1

Quiz threshold	Accuracy (%)	Execution time (s)
0	94.61	204.189
1	98.52	251.492
2	100	253.504
3	100	253.988

We obtained our experimental results, as shown in Table 2, when we set the malicious ratio to 0.3.

Table 2. Malicious ratio is set to 0.3

Quiz threshold	Accuracy (%)	Execution time (s)
0	82.53	204.313
1	90.58	252.462
2	94.74	255.259
3	97.43	256.657
4	100	261.151
5	100	264.739

Table 3. Malicious ratio is set to 0.5

Quiz threshold	Accuracy (%)	Execution time (s)
0	72.37	205.064
1	87.32	255.467
2	87.43	256.029
3	90.39	256.798
4	92.58	261.030
5	95.92	263.246
6	100	266.149
7	100	267.320

We obtained our experimental results, as shown in Table 3, when we set the malicious ratio to 0.5.

As seen in Tables 1–3, accuracy increased when the quiz threshold increased. When the quiz threshold was set to 6, the accuracy reached 100% in each simulation experiment. That is to say, when the quiz threshold was set to 6, we detected all the collusive workers without having to consider the malicious ratio. These results are similar to the results shown in Fig. 3.

In the three simulation experiments, we got the overhead with a different malicious ratio, as shown in Fig. 4. The overhead with a different malicious ratio was no larger than 2.5, and the overhead appeared to be stable.

We also obtained the execution time results. The execution time increased when the quiz threshold increased. In the following simulation experiment, the malicious ratio was set to 0.5. We compared the execution time difference between the proposed framework and the traditional MapReduce, as shown in Figs. 5 and 6.

Fig. 5 shows the comparison of response times when the traditional MapReduce and the proposed framework executed the same number of map tasks. Fig. 6 shows the comparison of response times when the traditional MapReduce and the proposed framework executed the map tasks with the same data size.

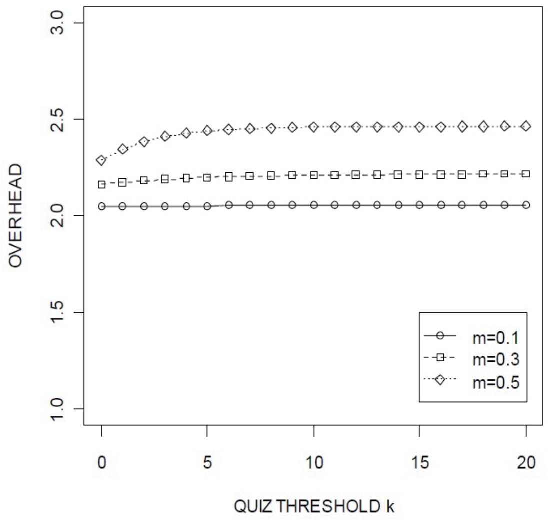


Fig. 4. The overhead with different malicious ratio.

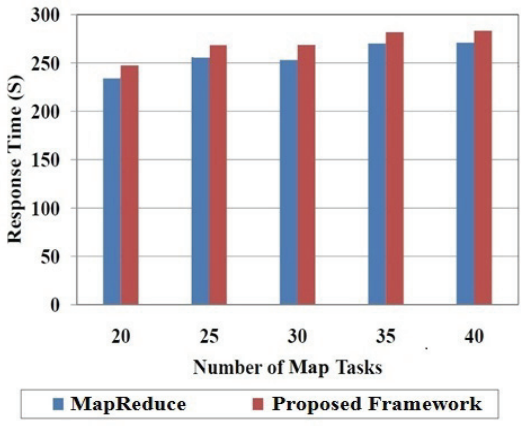


Fig. 5. The comparison with same number of tasks.

From the two response time comparisons above, we were able to get the response time for the proposed framework, and it was about 3% longer than the response time of the traditional MapReduce. Although the response time of the proposed framework was longer than the response time of the traditional MapReduce in the two comparisons, the response time differences were not significant. We considered the response time differences to be within the acceptable limits.

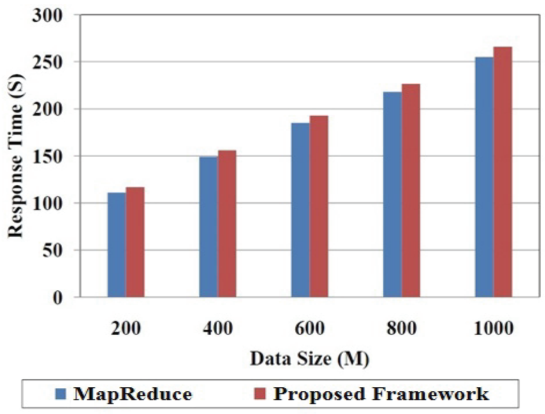


Fig. 6. The comparison with same data size.

5. Conclusions

In this paper, we have proposed a novel framework that overlay MapReduce for cloud computing to offer high integrity results. Using this framework, we were able to detect all of the collusive workers with different malicious ratios when we set the quiz threshold to 6. The overhead of the proposed framework was no larger than 2.5, even though the malicious ratio was 0.5. The response time of the proposed framework was about 3% longer than the response time of traditional MapReduce. So our theoretical analysis and the experiment results show that this framework can achieve high security for cloud computing without malicious workers. Based on our framework, we were able to accurately detect the non-collusive worker and collusive worker, respectively. This proves that our proposed framework is available, stable, and efficient.

We also noticed that the quiz threshold is important to accuracy. In this work, we set the quiz threshold by using artificial methods. We will keep on improving this framework so that the quiz threshold can be reasonably and automatically set on its own.

Acknowledgement

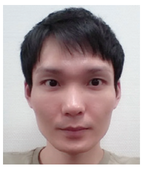
This research would not have been possible if we had not been provided with the opportunity to use the computer lab at Konkuk University, Seoul. Therefore, we would like to sincerely thank all of the staff and lecturers who helped conduct this research and who made this paper possible.

References

[1] R. Lammel, "Google's MapReduce programming model-revisited," *Science of Computer Programming*, vol. 70, no. 1, pp. 1-30, 2008.

[2] S. N. Srirama, P. Jakovits, and E. Vainikko, "Adapting scientific computing problems to clouds using MapReduce," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 184-192, 2012.

[3] F. Costa, L. Silva, and M. Dahlin, "Volunteer cloud computing: MapReduce over the Internet," in *Proceedings of 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW),* Shanghai, China, 2011, pp. 1855-1862.
[4] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29-43, 2003.
[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.
[6] Apache [Online]. Available: http://hadoop.apache.org.
[7] Hadoop [Online]. Available: http://hadoop.apache.org/releases.
[8] W. Wei, J. Du, T. Yu, and X. Gu, "SecureMR: a service integrity assurance framework for MapReduce," in *Proceedings of Annual Computer Security Applications Conference (ACSAC'09)*, Honolulu, HI, 2009, pp. 73-82.



Wenzheng Zhu http://orcid.org/0000-0001-7152-9049

He received his B.S. degree in the Department of Computer Science from Wonkwang University, Korea, and his M.S. degree in the Department of Computer Science from Konkuk University, Korea. He is currently working towards Ph.D. degree in the Department of Computer Science at Konkuk University. His research interests include Cloud Computing, OS and Information Security.

Changhoon Lee http://orcid.org/0000-0002-5249-1695

He is a professor in the Department of Computer Science at Konkuk University. He received his B.S. degree in the Department of Mathematics from Yonsei University, Korea, and his M.S. and Ph.D. degree in the Department of Computer Science from KAIST, Korea. His research interests are in the areas of AI, OS and Information Security.