VIAF: Verification-based Integrity Assurance Framework for MapReduce

Yongzhi Wang
Florida International University
Miami, FL, USA
ywang032@cs.fiu.edu

Jinpeng Wei
Florida International University
Miami, FL, USA
weijp@cs.fiu.edu

***Abstract*—MapReduce, a cloud computing paradigm, is gaining popularity. However, like all open distributed computing frameworks, MapReduce suffers from the integrity assurance vulnerability: it takes merely one malicious worker to render the overall computation result useless. Existing solutions are effective in defeating the malicious behavior of *non-collusive* workers, but are futile in detecting *collusive* workers. In this paper, we focus on the *mappers*, which typically constitute the majority of workers, and propose the Verification-based Integrity Assurance Framework (VIAF) to detect both *non-collusive* and *collusive* mappers. The basic idea of VIAF is to combine task replication with non-deterministic verification, in which consistent but malicious results from collusive mappers can be detected by a trusted *verifier*. We have implemented VIAF in Hadoop, an open source MapReduce implementation. Our theoretical analysis and experimental result show that VIAF can achieve high task accuracy while imposing acceptable overhead.**

Keywords- MapReduce; Result Verification; Collusion detection; Hadoop

I. INTRODUCTION

Ever since the genesis of cloud computing, discussion about its security has never stopped. On one hand, as the evolution outcome of web application, cloud computing, by nature inherited existing threats such as phishing [1], downtime [2], data loss [3], password weaknesses [4], and compromised hosts running botnets [5]. On the other hand, with unique characteristic, it brings new challenges such as unexpected side channels and covert channels [6]. Of all the issues threatening cloud computing, computation integrity is one of the most critical that needs attention. Due to the distributed architecture and open environment, some participants can be subverters that pretend to be good but actually perpetrate cybercrime or other cyber attacks [7].

In our research, we found MapReduce, a popular cloud computing framework, to a great extent, suffers from such integrity vulnerability. Since computation job is carried out via the collaboration of a number of computing nodes, which may not be trusted in an open environment, merely one corrupt result may render the overall result useless. In MapReduce, reducers normally constitute the minority of workers, so they can be deployed on trusted nodes to achieve high computation accuracy. However, it is often infeasible to deploy mappers on trusted nodes due to their large quantity. Hence, ensuring mappers' computation integrity is of great importance and is in urgent need of a solution.

Several existing techniques such as replication, sampling, and checkpoint-based solution have been proposed in the hope of addressing such issue in many distributed environments such as P2P Systems, Grid Computing and Cloud Computing [8-15]. However, since the cloud computing normally is applied to process critical data such as scientific computation and commercial data mining, above methods either requires too much overhead in order to guarantee high accuracy or can only deal with naïve attackers such as non-collusive attackers. For example, SecureMR [15] is purely based on replication, thus it cannot detect collusive workers that can cooperate to hide their attacks, i.e., returning consistent but bad results. For the quiz-based idea proposed in [9] used for P2P grid, when the percentage of malicious hosts increases from 5% to 50%, the overhead increases quickly from 2 to 4.5. Moreover, the idea proposed in [9] is to insert quizzes into a computation package, which cannot be directly applied to MapReduce.

Based on the idea of replication-based and quiz-based method, we propose VIAF (Verification-based Integrity Assurance Framework), a MapReduce framework that can detect both collusive and non-collusive mappers and thus guarantee high computation accuracy. In VIAF, we replicate each mapping task to detect non-collusive mappers. In addition, we add limited number of trusted computation nodes called *verifier* to verify a small portion of consistent results in a random manner and thereby detect collusive mappers. We call passing one verification as passing one quiz. We accumulate the number of passed quizzes for each computation node. If a node passes certain amount of quizzes, we believe that this node is not a malicious one and accept its result. Once a node fails any quiz, we confirm that it is a malicious node and add it to a blacklist.

We perform theoretical analysis about VIAF and implement it on top of Apache Hadoop MapReduce[16]. Both theoretical analysis and experimental result demonstrate that VIAF can achieve high accuracy while incurring acceptable overhead.

The rest of the paper is organized as follows. Background knowledge and the system model are presented in Section II. System design and theoretical analysis is described in Section III. System implementation, experiment result and analysis are discussed in Section IV. Section V discusses related work. Finally, Section VI gives the conclusion and future work.

II. BACKGROUD AND SYSTEM MODEL

A. MapReduce

MapReduce [17] is a framework of performing data-intensive computations in parallel on commodity computers. In MapReduce, each computation request issued by the user is called a *job*. Each job is usually broken down into several *tasks*. Once all the tasks are finished, the job is completed. The traditional architecture of MapReduce consists of one *master* and a number of *workers*. 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. In addition to the master and workers, the *Distributed File System (DFS)* is also an important component of MapReduce for data storage. At the beginning of computation, input data will be retrieved from the DFS and the result data will be stored to the DFS after the job is finished.

An ordinary MapReduce computation process can be divided into two phases: *map* and *reduce*. In the map phase, input data from the DFS are divided into several chunks. Each data chunk will be assigned to one worker as a task input to compute independently. And the computation result will be written to the worker's storage temporarily for future use. In this phase, the task that each worker is performing is called a *mapping task*. Those workers are called *mappers*. The map phase is followed by the reduce phase. In the reduce phase, one or several workers aggregate the intermediate result into a complete result according to some function and write the result into the DFS. Similarly, the task being executed is called a *reducing task* and the workers are called *reducer*.

B. System model and Assumptions

The MapReduce system discussed in this paper is implemented to run in an open system, where entities (DFS, master, workers) may come from different trust domains. Hence not all entities are trusted. In our system model, we assume the master and the DFS are trusted, where workers are not. Our goal in this paper is trying to ensure high accuracy of computing using untrusted workers. In this paper, we introduce a new type of worker called *verifier*, which is responsible for verifying mapper's computation result in order to guarantee the overall computation accuracy. Therefore, our system has three types of workers: mapper, reducer and verifier. Since reducers and verifiers typically take the minority portion among workers, we assume they are running on the trusted nodes. Therefore, the only untrusted nodes are mappers, which, however, take the majority portion. We assume the number of benign mappers in the cloud environment is dominating the number of malicious mappers. Also, since our framework asks each mapper to report its intermediate result (in the form of hash code) to the masters but keep the real result data in its local storage for the future reduce, we assume the result reported to the master is consistent with the local storage. This can be guaranteed by a commitment-based protocol such as SecureMR[15]. Based on the above assumption, the rest of our discussion only focuses on the accuracy of the map phase.

In addition, we only focus the information integrity on whether the computation node could provide correct calculation result, without considering the integrity vulnerability on the network. In other words, we assume the network facility is trusted.

C. Attack Model

The attackers in this system model are actually the malicious workers that try to generate bad result in order to sabotage the job output. They can be categorized into two types: *Non-collusive malicious worker* and *Collusive malicious worker*. (In the rest of this paper, we simply call them *non-collusive worker* and *collusive worker,* respectively.) A non-collusive worker normally returns bad result without consulting other malicious workers. In this case, if the same task is assigned to two workers and at least one of them is non-collusive, the cheat behavior can be easily detected by comparing the returns. In contrast, collusive workers can communicate with each other before cheating. When a collusive worker is assigned a task, it normally consults its collusive partner to see if they are assigned the same task. If yes, they will return consistently bad result; otherwise, they just return correct result. Since collusive workers try their best to minimize the inconsistency of their returns, they are much harder to detect than non-collusive workers.

III. SYSTEM DESIGN AND ANALYSIS

A. System Design

The basic idea of VIAF is a combination of task replication and non-deterministic result verification. In VIAF, each task is assigned to two workers, and the results are required to return. The consistency of two workers' returns will be checked in order to detect non-collusive workers. In addition, we add a type of trusted worker, the *verifier*, to the VIAF to verify the correctness of consistent returns, in order to detect collusive workers. Each worker has to accumulate its credit by passing enough verification. Task results are first stored locally by each worker. When a worker accumulated enough credits, all its stored results will be accepted by the master. By using replication and result verification, we can efficiently defeat both non-collusive workers and collusive workers. Fig. 1 depicts the data flow of the VIAF. In Fig. 1, the Task Queue, History Caches for each worker and the Result Buffer are maintained in the master. w1 and w2 are any two workers performing the same map task. The Verifier and the Reducer, as mentioned in Section II.B, are trusted entities. With Fig. 1, we can easily describe the control flow of the VIAF.

Whenever the Task Queue is not empty, the master will pick one task from it and send it to any two workers, shown in step 1 of Fig. 1. After calculation, both workers will return

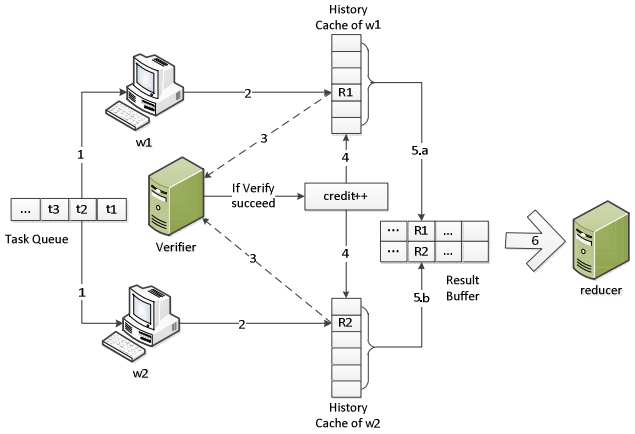

Figure 1. Data flow of VIAF

their results (in the form of hash code) to the master. If the returns are different, the master will reschedule the task to two new workers. (In this case, the master realizes that at least one worker is a non-collusive worker, but it is hard to tell which one is. So it has to reschedule the task to two different workers in the hope that the newly assigned workers are not malicious.) If the returns are the same, the master caches the two workers' result and task information to their History Caches respectively (step 2). Those cached information are called *history*. After that, the master might ask the verifier to verify this consistent result (step 3). Due to the limited resource of the verifier, verification will be launched in a non-deterministic manner (with certain probability). If the verification fails (the verifier returns a different result), the master is confirmed that the two workers are collusive workers. They will be added to a black list. All the results in their History Caches are untrusted and all the tasks in their History Caches will be rescheduled. On the contrary, if the verification succeeds (the verifier returns the same result as the workers), we say each of the two workers has passed one *quiz* or accumulated one *credit*. Their credit will be incremented respectively (step 4). Each worker, after accumulating adequate credits, will obtain the master's trust, and all the results cached in its History Cache will be released to the Result Buffer (step 5.a and 5.b). Since the credit of each worker is accumulated independently, the release of Cache History for each worker happens independently. Since the two workers are chosen in a non-deterministic manner (the master only guarantees that replicate tasks are not assigned to the same worker), it is very likely that when one verification successfully increases two workers' credit, one has accumulated enough credit and can release its History Cache to the Result Buffer, whereas the other still hasn't accumulated enough credit. After releasing the result, the worker's credit is reset to 0 and its History Cache is cleared. Since our strategy is to release the result to the reducer only when both workers executing the task are trusted, the Result Buffer is therefore designed to buffer the result of the replicate tasks. Only when two results of one task are received by the Result Buffer, can the master release it to the reducer (step 6). In other words, the result received by the reducer is issued by two workers, both of which have passed adequate quizzes and obtained the master's trust. If one worker fails to pass a quiz, not only will the tasks in its History Cache be rescheduled, but also the relevant replicate tasks already received by the Result Buffer will be rescheduled. The algorithm can be expressed as Fig. 2.

In the algorithm of VIAF, we declare the number of credits a worker needs to accumulate in order to earn the master's trust as *quiz_threshold*. Each worker's History Cache and credit are initialized in the function *Initialization()*. After initialization, the function *Task_scheduler()* keeps on monitoring the Task Queue to schedule the task and accumulate the credit for each worker. When a worker has accumulated enough credit, the master will release its History Cache to the Result Buffer. Meanwhile, the function *ResultBuffer_monitor()* keeps on monitoring the Result Buffer and release the result to the reducer once both results of replicate tasks are received by the Result Buffer.

Although the algorithm depicted in Fig. 2 is straightforward, some details are skipped for clarity. For example, when a task is released or rescheduled, the corresponding task information in its History Cache should be cleared. Since verification result is trustworthy, it can be directly released to the reducer without being added to the History Cache. Since each task is replicated, releasing results to the reducer should avoid passing the duplicate results. To achieve randomized verification effect, each time when the consistent duplicate results are received by the master, the master will verify the result with certain probability. We call such probability the *verification probability*. Apparently, the verification probability dictates the workload of verifiers. Finally, although verification is launched in a randomized way, it is also necessary to guarantee the verification is balanced across all the workers. Otherwise, some workers may cache too many results before releasing them.

For the non-collusive workers, since their bad returns will always be detected, their task will always be rescheduled. Therefore, non-collusive worker will not undermine the accuracy of job result. For the collusive workers, since the verification is invoked in a random way, they cannot predict whether their return will be verified. The only way to avoid detection is to return correct result. For those collusive workers who try to take a risk to evade verification, the probability to pass a series of quizzes turns to be very low. Passing more quizzes makes the worker to be more trusted, but it requires more space to buffer the intermediate result and delay the overall computation time. Therefore, choosing a proper *quiz threshold* (the number of quizzes that a worker has to pass in order to obtain trust) becomes critical. The analysis below indicates the relationship between quiz threshold and accuracy requirement.

B. Theoretical Analysis

Our analysis model assumes a cloud environment that contains a large number of workers, specifically, mappers. Suppose the environment contains N workers, and M out of which are malicious, we define malicious node ratio m = M/N. For simplicity, we assume that m stays constant throughout the job execution. That is, even though some

```
const quiz_threshold;

Initialization(){
 for each worker(w){
  //tasks executed but not released by w
  w.tasks = {};
  //results generated but not released by w
  w.results = {};
  //quizzes w has passed
  w.credit = 0;
 }
}

Task_Scheduler(){
 while (task queue Q is not empty){
  select a task t from task queue;
  assign t to any 2 workers w1 and w2;
  receive result R1,R2 from w1,w2;
  if(R1 != R2) // inconsistent result
   put t back to Q; //reschedule
  else{ // consistent result
   add t to w1.tasks and w2.tasks;
   add R1 to w1.results and w2.results;
   with probability (p){
    verify t and receive result Rv;
    if(Rv== R1){
     w1.credit++;
     w2.credit++;
     if(w1.credit>quiz_threshold){
      release w1.results to resultBuffer;
      reset w1.tasks,w1.results, w1.credit;
     }
     if(w2.credit>quiz_threshold){
      release w2.results to resultBuffer;
      reset w2.tasks,w2.results, w2.credit;
     }
    }
    else{
     add w1, w2 to the black list;
     put w1.tasks, w2.tasks back to Q;
     find the replicate tasks of w1.tasks and
w2.tasks that are released to the resultBuffer,
remove their results from resultBuffer and
reschedule them;
    }
   } // end with probability p
  } // end consistent result
 } // end while
}

ResultBuffer_monitor(){
 for each task t in resultBuffer{
  if(two results are released from workers)
     release the result to reducer;
 }
}
```

Figure 2. The algorithm of VIAF

malicious nodes are detected and black-listed during job execution, m decreases very little given the large values of N and M.

We also assume M malicious nodes include both workers: collusive and non-collusive workers. We define c as the portion of collusive worker out of M malicious workers. Similarly, we also assume c as a constant parameter due to the large value of M. Even though a task is assigned to two collusive workers, collusion can happen only when they are in the same collusion group (in some environment, there may exists several groups of collusive workers, collusion can only happen within a group) and they are executing the task simultaneously (for example, collusion cannot happen if the two workers fall out of sync in processing the same task). Due to the uncertainty of temporal sequence, we define p as the probability that one collusive worker can find its partner executing the same task and be able to commit a cheat. When the collusive workers find each other, they can cheat in a non-deterministic way in the hope of not been discovered very soon. We define q as the probability that the collusive partners decide to commit a cheat. Similarly, we define r as the cheat probability of non-collusive workers. As mentioned in Section III.A, verification is launched with a certain probability, called the *verification probability*. We denoted it as v. Also, we denote the quiz threshold as k.

With the model defined above, we are about to analyze the following measurement metrics:

Definition III.1 (Survival Chance) The Survival Chance of a collusive worker is the probability that it passes all the k quizzes and earns trust of master so that the result of its History Cache can be released to the Result Buffer. It is denoted as Δ.

Definition III.2 (Cheat Probability) The Cheat Probability of a task is the probability that the master accepts a bad result and releases it to the reducer, denoted as *CP*.

Definition III.3 (Accuracy) The Accuracy of a task is the probability that the master accepts a good result and releases it to the reducer, denoted as *AC*. Apparently, AC = 1-CP.

Definition III.4 (Overhead) The Overhead of a task is the average number of execution launched by the worker before its result is released to the reducer. It includes task replication and task reschedule. It is denoted as *OH*.

Definition III.5 (Verification Overhead) The Verification Overhead of a task is the average number of execution launched by the verifier before its result is released to the reducer. It only includes task verification. It is denoted as *VO*.

The model parameter and measurement metrics are summarized in TABLE I.

Let's start with the analysis of survival chance Δ. In our analysis model, we don't eliminate non-collusive workers because they cannot affect the correctness of job result, due to the existence of replication. So the survival chance of non-collusive workers is 1. For benign workers, the survival chance is always 1, since they never return bad result. Now, let's consider the survival chance Δ of each collusive worker. For a collusive worker, passing one quiz must happen in one of the four scenarios below:

a. The replicate task is also assigned to another collusive worker (with probability m*c), they are able to collude (with probability p), but they determine not to (with probability 1-q). So the probability in this case is A = mcp(1-q);
b. The replicate task is assigned to another collusive worker (with probability m*c), but they cannot collude either because they belong to different collusive group or they are not executed simultaneously (with probability 1-p). The probability in this case is B = mc(1-p);

TABLE I. NOTATION FOR THEORETICAL ANALYSIS MODEL

Symbol	*Explanation*
m	Malicious worker ratio out of all workers
c	Collusive worker ratio out of malicious workers
p	When a task is assigned to two collusive workers, the probability that two workers are in one collusive group and can discover each other, which is to say, they are able to commit a cheat.
q	The probability that two collusive workers determine to commit a cheat when discovering each other.
r	The probability that a non collusive worker determine to commit a cheat when assigned a task.
v	(Verification Probability) The probability that a task returning consistent results is verified by the verifier.
k	(Quiz Threshold) Number of quizzes a worker must passed in order to obtain the trust of the master.
Δ	(Survival Chance) The probability that a worker passes all the k quizzes and earns trust of master.
CP	(Cheat Probability) The probability that a task returns a bad result to the master and the master releases it to the reducer.
AC	(Accuracy) The probability that a task returns a good result to the master and the master releases it to the reducer.
OH	(Overhead) The average number of execution launched by the worker for each task.
VO	(Verification Overhead) The average number of execution launched by the verifier for each task.

c. The replicate task is assigned to a non-collusive worker (with probability m(1-c)), and it does not commit a cheat(with probability 1-r). The probability is C = m(1-c)(1-r);
d. The replicate task is assigned to a benign worker. The probability is D = 1-m.

To survive, the k passed quizzes must be distributed among the four scenarios above. Without losing generality, for any case in which scenario a, b, c and d happens i, j, h and k-i-j-h times, respectively, the probability of each combination in this case is $A^i B^j C^h D^{k-i-j-h}$. For this case, there are $\binom{k}{i}\binom{k-i}{j}\binom{k-i-j}{h}$ combinations, so the probability of this case is

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

By permutation, the survival chance of a collusive worker Δ is therefore the summation probability of all possible cases:

$$\Delta = \sum_{i=0}^{k}\sum_{j=0}^{k-i}\sum_{h=0}^{k-i-j} P(i,j,h,k) \quad (1)$$

With the survival chance, we can derive Cheat Probability CP. The master releases a bad result to the reducer only in one of the following three scenarios:

a. The task is assigned to two collusive workers. They both survive in k quizzes and they commit a collusive cheat. The probability in this case is $m^2c^2pq\Delta^2$.
b. The task is assigned to two collusive workers. Any of them fails to pass the k quizzes, so the task has to be rescheduled. Knowing the Cheat Probability of a rescheduled task as CP, we have the cheat probability as $m^2c^2(1-\Delta^2)$CP in this case.
c. The task is assigned to at least one non-collusive worker and it commits a cheat. The probability of this case is m(1-c)r. In this case, the task should be rescheduled, so the probability is m(1-c)rCP.

Combining the three cases, we have the Cheat Probability CP as:

$$CP = m^2c^2pq\Delta^2 + m^2c^2(1-\Delta^2)CP + m(1-c)\cdot r \cdot CP$$

Therefore,

$$CP = \frac{m^2c^2pq\Delta^2}{1-m(1-c)r-m^2c^2(1-\Delta^2)} \quad (2)$$

Since *AC = 1-CP*, we have

$$AC = 1 - \frac{m^2c^2pq\Delta^2}{1-m(1-c)r-m^2c^2(1-\Delta^2)} \quad (3)$$

Overhead OH can be calculated with the same principle. In our model, rescheduling happens in three cases:

a. The task is executed by two collusive workers and one of them fails to pass the k quizzes. The probability of this case is $m^2c^2(1-\Delta^2)$. The overhead is 2+OH, where 2 is for the replication and OH is the overhead caused by rescheduling.
b. The task is assigned to at least one non-collusive worker and it commits a cheat. The probability is m(1-c)r. Similarly, the overhead is 2+OH.
c. For the other cases, the overhead is 2 since there's no reschedule needed. The probability is $1-m^2c^2(1-\Delta^2)-m(1-c)r$.

Adding above overheads together, we have

$$OH = m^2c^2(1-\Delta^2)(2+OH) + m(1-c)r(2+OH) + 2(1-m^2c^2(1-\Delta^2)-m(1-c)r)$$

Therefore, we have derived the Overhead:

$$OH = \frac{2}{1-m(1-c)r-m^2c^2(1-\Delta^2)} \quad (4)$$

Similarly, we can calculate the verification overhead. Suppose the average verification overhead of each task is VO. Consider the following cases:

a. The task is assigned to two collusive workers and one of them fails to pass the k quizzes. The probability in this case is $m^2c^2(1-\Delta^2)$.When the task is verified (with probability v), the verification result will be directly accepted by the master. The verification overhead is $m^2c^2(1-\Delta^2)v$. However, when the task is not verified (with probability of 1-v), it will be rescheduled. Since after reschedule, the average verification overhead is VO, we have the verification overhead as $m^2c^2(1-\Delta^2)(1-v)$VO. Combining the two cases, we have the verification overhead $m^2c^2(1-\Delta^2)v+ m^2c^2(1-\Delta^2)(1-v)$VO.
b. The task is assigned to at least one non-collusive worker

and it commits a cheat. The probability in this case is m(1-c)r. Since task will be rescheduled in this case, the verification overhead is m(1-c)rVO.

c. For the other cases, where the probability is (1- $m^2c^2(1-\Delta^2)$-m(1-c)r), the task will not be rescheduled but can be verified with probability v. The verification overhead is therefore (1-$m^2c^2(1-\Delta^2)$-m(1-c)r)v.

The above three cases consists the verification overhead VO:

$$VO = m^2c^2(1-\Delta^2)v + m^2c^2(1-\Delta^2)(1-v)VO + m(1-c)r \cdot VO + (1-m^2c^2(1-\Delta^2) - m(1-c)r)v$$

Solving above equation, we have

$$VO = \frac{v(1-m(1-c)r)}{1-m(1-c)r-m^2c^2(1-\Delta^2)(1-v)} \quad (5)$$

Apparently, verification overhead is useful in calculating the workload of verifiers. Suppose a job consists of T tasks, the verifiers' workload is T*VO.

Fig. 3 shows the simulated relationship between quiz threshold and accuracy based on (3) and (1). In this figure, malicious node ratio is set to 0.1, 0.3 and 0.5, respectively. Half of the malicious workers are collusive, they commit collusive cheating with probability 0.5*0.5=0.25, and non-collusive workers commit inconsistent cheat with probability of 50%. We can see in this case, the accuracy increases rapidly when the quiz threshold increases from 1 to 6. When quiz threshold grows to 7, the accuracy reaches almost 100%. Fig. 4 and Fig. 5 demonstrate the overhead and verification overhead under the same configuration as Fig. 3. Here we set verification probability v to 0.2. We can see both the overhead and the verification overhead growth is very modest when the quiz threshold is increasing.

When the collusive worker ratio c increases to 1, where malicious workers are all collusive, malicious ratio m and probability p became important factor in determining the relationship of quiz threshold and accuracy. Fig. 6 demonstrates the relationship under different m. We notice that with a constant p, different m needs almost the same quiz threshold in order to achieve high accuracy. Fig. 7 demonstrates the relationship under different p. We observe that p determines the quiz threshold in a subtle manner: to achieve very high accuracy, the smaller the p is, the greater the quiz threshold is needed. For example, when p is 1.0, five quizzes can make the accuracy almost achieve 100%; when p is 0.3, more than 10 quizzes are needed to guarantee such a high accuracy. The intuition here is: when malicious workers are more likely to cheat (with larger p), accuracy suffers more without verification; at the same time however, they are easier to detect once verification is used. Thus verification tends to be more effective in improving the accuracy. On the other hand, when the malicious workers are more stealthy (cheating with smaller p), accuracy suffers less, but it is also harder to improve by verification. Fortunately, when the accuracy required is not very high, small number of quiz threshold will satisfy. For example, when the accuracy requirement is 98%, five quizzes will be enough for all three values of p. The overhead and the verification overhead corresponding to Fig. 7 are shown in Fig. 8 and Fig. 9, respectively. Still the overhead with different p is no larger than 2.5, and the verification overhead appears to be stable.

Based on the above analysis, we observe that p is an essential factor in determining the quiz threshold. When p is greater than 0.5, passing 7 quizzes will guarantee very high accuracy for each worker (close to 100%). When p is less than 0.5, to guarantee high accuracy, a higher quiz threshold is needed.

IV. IMPLEMENTATION AND EVALUATION

We have modified Hadoop MapReduce 0.21.0 to implement a prototype of VIAF. With this implementation, we did a series of experiment. The experimental results have been encouraging.

A. Implementation details

In Hadoop, each worker communicates with the master via "heartbeats". In each heartbeat, the worker reports task update information to the master and requests new task to execute. When a task is done, a *task complete event* is passed to the master. The master will add such events to a queue so that the reducer can fetch the result and continue with the reduce task. When a worker is requesting new tasks, the master will choose an unexecuted task from its queue and assign it to the worker.

In our implementation, we modify Hadoop to implement the algorithm of VIAF discussed in Fig. 2. Each worker has to return MD5 hash code of the task result along with the task completion event. Meanwhile, instead of directly adding the events to the queue, we cache the event of each task until both workers who submit the replicate task result have passed k quizzes. Also, we modify the task assignment

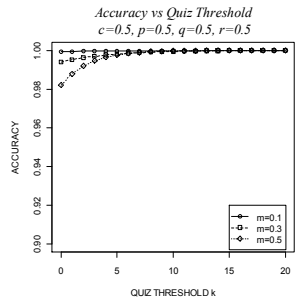

Figure 3. Accuracy for miscellaneous workers

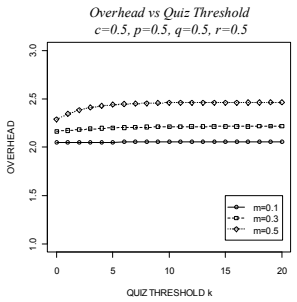

Figure 4. Overhead for miscellanious workers

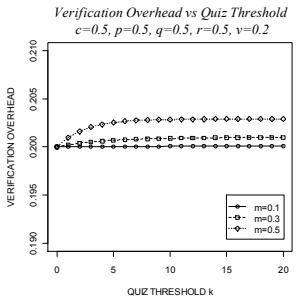

Figure 5. Verification Overhead for miscellaneous workers

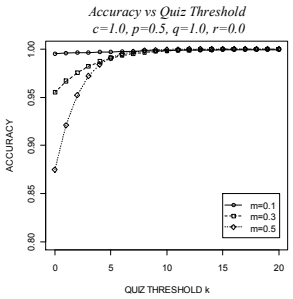

Figure 6. Accuracy with different m

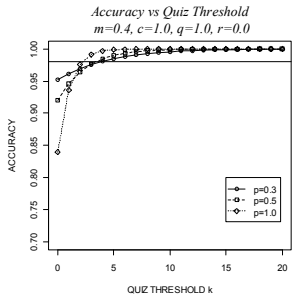

Figure 7. Accuray with different p

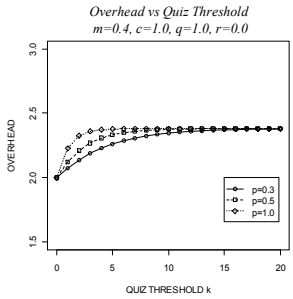

Figure 8. Overhead with different p

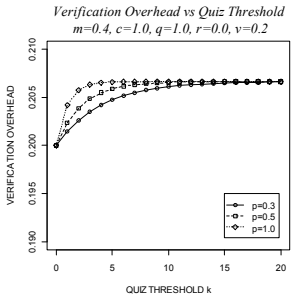

Figure 9. Verification Overhead with different p

function. Instead of randomly assigning the task, we guarantee that replicated and rescheduled tasks are not assigned to the same worker and that the verifier is only assigned verification tasks.

B. Collusive worker model

In our experiment, we design the behavior of collusive worker as follows: a collusive worker functions as both a client and a server. As a client, when a task is assigned, it will contact all its collusive group members to see if anyone is running the same task. If getting affirmative response from any partner (a server worker), they will negotiate whether to cheat (actually, whether to cheat is determined by the server worker, in our implementation, the collusion probability q is set to 1). If not getting response, it will execute the task in a benign way. Meanwhile, it will function as a server to monitor if any other collusive worker contacts it and claims running the same task. Once received such a message from any client worker, it will determine whether to cheat and instruct the client worker. Since the server worker usually execute the task before the client worker, to keep consistent return, the server worker usually needs to abort the current work and start to generate a consistent bad result when receiving a message.

C. Experiment and result analysis

We launch our experiments on a 2.93 GHz, 8-core Intel Xeon CPU with 16 GB of RAM running VMware Workstation 7.11. We deployed 11 virtual machines (512MB of RAM and 40GB of disk each) to construct a MapReduce environment. Each machine runs on Debian 5.0.6 "lenny". Out of the 11 nodes, one is running as both a master and a benign worker; one is running as a verifier worker; four are collusive workers in one collusive group implementing the model in Section IV.A; and the remaining five nodes are benign workers. All experiments use the Hadoop WordCount application [18]. The input files are text files downloaded from a free eBook project website [19]. The complete job requires 400 mapping tasks and one reducing task with standard MapReduce.

TABLE II shows the accuracy and overhead with different quiz threshold. Here we set a verification probability of 20%. In TABLE II, accuracy represents the ratio of accepted correct results to overall accepted results (400). From the table, we can see that without quiz test, accuracy is only 87.2%, but with quiz tests the accuracy rapidly improves. For example, a Quiz Threshold of 1 results in a 99.42% accuracy.

Since our experiment environment only has collusive workers, c is effectively 1.0. Besides, our implementation of collusive workers sets p and q close to 1.0. Hence, the experiment configuration at the beginning is close to the theoretical model in Fig. 7. But we find the experiment result is much better than the Fig. 7 in terms of accuracy. Such discrepancy can be explained with Fig. 6: when other parameters are constant, the smaller the m is, the higher accuracy it will achieve. In our experiment environment, collusive workers are eliminated after detection, which makes the malicious ratio m decrease from 0.4 towards 0. Hence, the accuracy presented in Fig. 7 is only a lower bound. Similarly, with the decrease of m, the overhead decreases. This explains why the experiment overhead is lower than the simulated results in Fig. 8.

Introduction of verification in VIAF will delay the job completion time, but our experiment shows that such delay is

TABLE II. EXPERIMENT RESULT WITH DIFFERENT QUIZ THRESHOLD

Quiz Threshold	*Accuracy*	*Overhead*	*Verification Overhead*
0	87.20%	2.000	0
1	99.42%	2.045	22.00%
2	99.83%	2.074	23.58%
3	100%	2.053	23.00%
4	100%	2.162	23.58%
5	100%	2.046	21.75%
6	100%	2.111	22.58%
7	100%	2.027	19.83%

TABLE III. JOB EXECUTION TIME FOR DIFFERENT QUIZ THRESHOLD

Quiz Threshold	*Execution time (s)*	*Execution time increase compared with no quiz (%)*
No quiz	429.070	---------
1	475.064	10.72%
2	472.426	10.10%
3	473.149	10.27%
4	476.798	11.12%
5	460.029	7.22%
6	483.246	12.63%
7	473.030	10.25%

acceptable. According to the collusive worker model described in Section IV.B, the collusive workers may take longer time to return results compared to the benign workers. However, delays due to such worker-specific behaviors can be arbitrary and are not part of the overhead introduced by our framework. So, our experimental environment only includes benign workers in order to rule out the delay caused by malicious workers. TABLE III shows the job execution time under different Quiz Threshold. Compared with the case in which no verification is applied, we can see that introducing verification adds a small amount of job completion delay. But such delay appears to be unrelated to the quiz threshold setting.

V. RELATED WORK

A number of techniques such as replication, sampling, and checkpoint-based solution have been proposed to address cheating in several distributed computing contexts, such as P2P systems, Grid Computing, and Cloud Computing [8-15]. Quiz-based schemes such as [9] insert quiz tasks with verifiable results to detect malicious workers. In order to generate quizzes that are indistinguishable from normal tasks, [20] proposes to use tasks of the given jobs themselves as quiz questions. [21] employs a replication-based scheme that allows the degree of redundancy to be adaptively adjusted based on the dynamically-calculated reputation as well as reliability of each worker.

Similar to VIAF, the LLFT middleware [22] for a cloud computing or data center environment employs replication techniques. However, LLFT is designed for tolerating fail-stop faults for distributed applications, thus it cannot handle Byzantine faults such as the collusion among malicious workers in MapReduce.

VI. CONCLUSION AND FUTURE WORK

To guarantee high accuracy of MapReduce calculation, we propose the VIAF (Verification based Integrity Assurance Framework) to defeat both collusive and non-collusive malicious mappers. We present the design and implementation detail of such a framework, along with theoretical analysis and experimental evaluations. Our analysis and evaluations confirm that this framework can ensure high accuracy while incurring acceptable overhead and delay.

Although VIAF is a promising method to ensure MapReduce computation integrity, it can be further improved in the following directions. First, this paper is based on the assumption that the reducer is trusted; how to utilize VIAF to guarantee integrity without this assumption remains an interesting question. Second, since verifiers are precious resource, we can alleviate its workload further. For example, we can cache the results of verification and reuse verified tasks to test untrusted workers; then the verifier does not have to re-compute the reused tasks.

VII. ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Homeland Security under grant Award Number 2010-ST-062-000039. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

REFERENCES

[1] Gone phishing. Twitter Blog. January 03, 2009.
[2] E. Knorr. Gmail follies and Google's enterprise pitch. InfoWorld. September 8, 2009.
[3] J. Stokes. T-Mobile and Microsoft/Danger data loss is bad for the cloud. Ars technica. October 2009
[4] D. Raywood. The twitter hacking incident last week should be a call to better security awareness and not about cloud storage. SC Magazine. July 20, 2009.
[5] M. C. Ferrer. Zeus in-the-cloud. CA Community Blog. Dec. 9, 2009
[6] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploringinformation leakage in third-party compute clouds. In Proceedings of CCS'09.
[7] Y. Chen, V. Paxson, and R. Katz.What's New About Cloud Computing Security? Technical Report UCB/EECS-2010-5, Berkeley, 2010.
[8] W. Du, J. Jia, M. Mangal, and M. Murugesan, "Uncheatable grid computing,"in Proceedings of the 24th International Conferenceon Distributed Computing Systems (ICDCS'04), Washington, DC.
[9] S. Zhao, V. Lo, and C. GauthierDickey, "Result verification and trust based scheduling in peer-to-peer grids," in P2P '05: Proceedings ofthe Fifth IEEE International Conference on Peer-to-Peer Computing.Washington, DC, USA.
[10] L. F. G. Sarmenta, "Sabotage-tolerance mechanisms for volunteercomputing systems," Future Generation Computer Systems,vol. 18, no. 4, pp. 561–572, 2002.
[11] C. Germain-Renaud and D. Monnier-Ragaigne, "Grid result checking,"in CF '05: Proceedings of the 2nd conference on Computing frontiers. New York, NY, USA: ACM, 2005, pp. 87–96.
[12] P. Domingues, B. Sousa, and L. Moura Silva, "Sabotage-tolerance andtrust management in desktop grid computing," Future Gener. Comput.Syst., vol. 23, no. 7, pp. 904–912, 2007.
[13] P. Golle and S. Stubblebine, "Secure distributed computing in acommercial environment," in 5th International Conference FinancialCryptography (FC). Springer-Verlag, 2001, pp. 289–304.
[14] P. Golle and I. Mironov, "Uncheatable distributed computations," inCT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology.London, UK: Springer-Verlag, 2001, pp. 425–440.
[15] Wei Wei, Juan Du, Ting Yu, Xiaohui Gu, "SecureMR: A Service Integrity Assurance Framework for MapReduce", in Proceedings of the 2009 Annual Computer Applications Conference.
[16] "MapReduce Tutorial". http://hadoop.apache.org/mapreduce/docs/current/mapred_tutorial.html
[17] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1), 2008.
[18] "word count example" http://wiki.apache.org/hadoop/WordCount
[19] "Project Gutenberg" http://www.gutenberg.org/wiki/Main_Page
[20] P.Varalakshmi, S. Thamarai Selvi, K.Anitha Devi, C.Krithika, RM.Kundhavai A Quiz-Based Trust Model with Optimized Resource Management in Grid Computer Systems Architecture Conference, 2008. ACSAC 2008. 13th Asia-Pacific.
[21] Sonnek, J., Chandra, A., Weissman, J.B.: Adaptive reputation-based scheduling on unreliable distributed infrastructures. IEEE Trans. Parallel Distrib. Syst. 18(11), 1551–1564 (2007)
[22] Wenbing Zhao, P. M. Melliar-Smith and L. E. Moser. Fault Tolerance Middleware for Cloud Computing, In 2010 IEEE 3rd International Conference on Cloud Computing, July 2010, pp 67-74.