PAPER Special Section on Data Engineering and Information Management

VAWS: Constructing Trusted Open Computing System of MapReduce with Verified Participants

Yan DING[†a)], **Huaimin WANG**[††], **Lifeng WEI**[†], **Songzheng CHEN**[†], **Hongyi FU**[†], ***Nonmembers***, ***and*** **Xinhai XU**[†], ***Member***

SUMMARY MapReduce is commonly used as a parallel massive data processing model. When deploying it as a service over the open systems, the computational integrity of the participants is becoming an important issue due to the untrustworthy workers. Current duplication-based solutions can effectively solve non-collusive attacks, yet most of them require a centralized worker to re-compute additional sampled tasks to defend collusive attacks, which makes the worker a bottleneck. In this paper, we try to explore a trusted worker scheduling framework, named VAWS, to detect collusive attackers and assure the integrity of data processing without extra re-computation. Based on the historical results of verification, we construct an Integrity Attestation Graph (IAG) in VAWS to identify malicious mappers and remove them from the framework. To further improve the efficiency of identification, a verification-couple selection method with the IAG guidance is introduced to detect the potential accomplices of the confirmed malicious worker. We have proven the effectiveness of our proposed method on the improvement of system performance in theoretical analysis. Intensive experiments show the accuracy of VAWS is over 97% and the overhead of computation is closed to the ideal value of 2 with the increasing of the number of map tasks in our scheme.

key words: *result verification, computational integrity, MapReduce, open system, integrity attestation graph*

1. Introduction

MapReduce [1] computing paradigm becomes to play an important role in data processing in various open computing environments based on cloud computing [2], volunteer computing [3], [4], P2P system [5]–[7] and so on. Even the intelligent mobile phones can be exploited to construct a mobile computing platform of MapReduce [8]. Nodes involved in an open MapReduce environment, however, might come from different trust domains, with the potential threat of malicious attackers or cheating motivation to gain benefits. The existence of such nodes in the MapReduce framework might compromise the accuracy of final results. Therefore, it's critical to build a trusted framework from untrusted underlying computing resources in open mass data processing environment.

Current studies in this area are mainly based on the redundant computing principal, that is to dispatch task to multiple replicas, and vote on results to obtain correct ones [9]. Such scheme, however, is insufficient to effectively prevent collusive attacks. If malicious workers from the same collusive group comprise most of the participants in replication computing, the same false result can be obtained. Based on this, VIAF introduced a trusted role known as the verifier into the MapReduce framework to defend collusive attack [10]. The false results provided by collusive nodes can be detected by sampling the voted result and re-computing. This method is able to solve the collusive attack at a certain probability but requires the absolute trustworthiness of the verifier. Besides, this verifier has to re-compute all sampled tasks, and the results of mappers in the framework cannot be submitted until they pass the verification. From the perspective of scalability, this centralized verifier for re-computation might be the potential bottleneck of parallel computing.

Efficient verification without centralized verifier raises great challenges to system design. First, it is critical to evaluate not only the trustworthiness of the computing result but also that of the computing workers. Otherwise, the negative influence of malicious workers plays a part throughout the computation. Second, identification of malicious workers is difficult, especially for the collusive attack mode. If collusive attack occurs, workers that failed in the majority votes cannot simply be identified as malicious. Third, for the scalability, consideration must be given to the efficiency of both attacker identification and job computation. The earlier the malicious workers are excluded, the better performance will be achieved. Meanwhile, the computation performance of whole job shouldn't be affected too much by the scheme.

To address these challenges, in this paper, we propose VAWS (Verification-based Anti-collusive Worker Scheduling), a method of deploying determined benign workers to construct a collusion-resistant trusted MapReduce framework over open resources without extra re-computation. Consider that mappers constitute the majority of workers, VAWS focus on verifying mappers and assigning reducers and the master in the trusted domain. Based on the duplication verification, malicious workers are detected by analyzing inter-node verification information, and excluded from subsequent scheduling. Our major contributions are summarized as follows:

(1) A Verification-based Anti-collusive Worker Scheduling (VAWS) is proposed. VAWS is able to assure the integrity of MapReduce execution with higher accuracy without extra re-computation.

Manuscript received July 10, 2013.
Manuscript revised November 14, 2013.

[†]The authors are with School of Computer, National University of Defense Technology, Changsha, HN, P.R. China.
[††]The author is with National Laboratory for Parallel & Distributed Processing, National Univ. of Defense Technology, Changsha, HN, P.R. China.
a) E-mail: yanding@nudt.edu.cn

DOI: 10.1587/transinf.E97.D.721

(2) A malicious mapper identification method based on Integrity Attestation Graph (IAG) is designed. Through the analysis of the maximum clique problem (MCP) on IAG, the malicious workers in both collusion and non-collusion strategies can be effectively identified.

(3) A verification-couple selection method based on the influence of malicious workers under IAG guidance is also proposed. This method achieves high efficiency of locating malicious workers by determining inconsistency among workers.

The basic idea of IAG is proposed in [11]. We made modification on its definition to fit for our proposed attack model. Furthermore, in order to accelerate the identification rate, we proposed some heuristics to schedule replication verification pairs. The IAG-based malicious worker identification and heuristics of verification-couple selection are not specific to MapReduce, they could also be applied to other replication-based verification system to detect malicious workers with a high detection rate.

The paper is organized as follows. Section 2 presents a system model of open MapReduce computing and a cheating model. Section 3 presents the design of the collusion-resistant trusted scheduling method for mappers. Section 4 presents the theoretical analysis and experimental evaluation as well as the comparison of the results with those obtained in related studies. Section 5 provides a review of related literature. Section 6 presents the conclusions and suggestions for future studies.

2. Background and System Model

2.1 MapReduce in Open System

MapReduce is a framework of parallel data processing model, consisting of a single master node and several worker nodes. The master is responsible for job management and task scheduling, while the workers perform tasks assigned by the master. MapReduce process can be divided into two phases: *map* and *reduce*. First, the input job is partitioned into m tasks that are independent of each other during the map phase. The master node dispatches these tasks to several worker nodes (*mappers*) to perform parallel map operations. The computational result in this phase is called the *intermediate result*. After map computation, all intermediate results are partitioned into different r parts. Every partition is assigned to a worker node to cast a reduce operation; this worker is called a *reducer*. In the reduce phase, each reducer reads a partitioned intermediate result from all necessary mappers and casts a reduce operation to obtain a final result.

The trustworthiness of participating nodes must be guaranteed to exploit open computing resources to build a MapReduce system. Given that the master and reducers typically constitute only a minority of workers, we assume that they operate on trusted nodes whose computing results need not be verified. However, many mappers are deployed to untrusted nodes; thus, certain measures must be implemented to guarantee the integrity of the computing results and to ensure that only trusted results can be committed to the reduce phase. The system model is shown in Fig. 1.

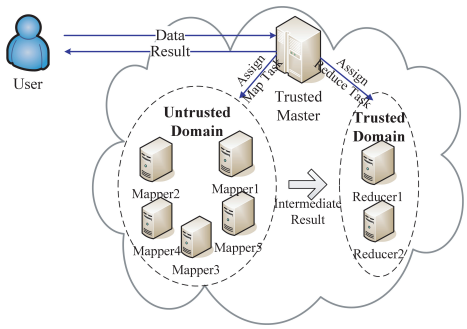


Fig. 1 System model of MapReduce in open system.

The common result verification method utilized on the untrusted mappers is replication and voting. The result can be committed to the next phase if and only if it wins the majority vote. Such a scheme only evaluates the trustworthiness of the computing result and not that of the computing workers. However, if malicious workers participate throughout the computing, their misbehaving results magnify the probability that inaccurate results are accepted as correct ones and induce more computing failure. So when designing the verification scheme of mappers, it is critical not only to protect the integrity of computation result, but also to identify the malicious workers and exclude their negative influence.

2.2 Attack Model

The attackers in this system model are the malicious workers supplying bad result to confuse the final output of the job. They can be categorized into two types: *non-collusive malicious workers* and *collusive workers*. Non-collusive malicious workers misbehave independently, while collusive ones may consult with each other and make an agreement. For example, when they are assigned the tasks with same input, they return the same results to avoid being detected even if they return wrong. Otherwise, if they return inconsistent results, their misbehavior turns to non-collusive mode, which is easier to be detected, making them exposed and the collusion meaningless. Thus, malicious workers from the same collusive group always supply the consistent results—that is, if they decide to misbehave, they may provide the same false result. Otherwise, both workers do not misbehave and return the correct result.

Assuming that there are N mappers, including m malicious ones ($\mathcal{A} = \{A_1, \ldots, A_m\}$), performing computing, each malicious worker A_i has a corresponding collusive group C_i which is a subset of $\mathcal{A}$. The possible patterns of attacker's behavior are categorized as below.

I. *Periodical Attackers*

For a dispatched slice in this attack pattern, malicious workers misbehave at fixed probability b ($0 < b \leq 1$). When b is 1, A_i always misbehaves. If the system dispatches a slice

to several replicas in this case, the malicious replicas could possibly misbehave through one of the following:

- *Non-collusive attack*: All replicas misbehave independently and offer false results with a probability of b.
- *Collusive attack*: Malicious workers from the same collusive group misbehave together with unique probability b, that is, they may provide the same false result if they decide to misbehave, otherwise, both workers do not misbehave. Malicious workers from different groups misbehave independently with a probability of b.

II. *Strategic Attackers*

In this pattern, the attackers previously know the replication strategy that how many replicas are used for verifying a task. And only when the collusive attackers are sure to hold over half number of all replicas and win the majority vote of result verification, do they misbehave.

For the duplication-based verification approach in our system designed in Sect. 3.1, both periodical attack and strategic attack must be prevented.

3. VAWS Design

3.1 Approach Overview

The main idea of VAWS is a combination of replication verification and malicious worker identification, as illustrated in Fig. 2. Duplication is utilized as a basic result verification mechanism. The master allocates two mappers for each data block to perform the computing. When the job is finished, mappers send the hash value of their intermediate results to the master. The master compares the two results. If the results are identical, then they can be trusted and will be copied by the reducer. Computing is considered a failure otherwise, and the other two mappers are scheduled to compute again. This procedure is repeated until the two results match.

According to the attack model defined in Sect. 2.2, given that the strategic attackers misbehave only if they would process all the two replicas, we can send the task to one replica first; the result from this replica must be obtained before sending the task to the other replica. In this way, the determination condition of strategic attacks is broken, and attacks are blocked. So the key difficulty is to defend the periodical attackers, especially in collusive mode.

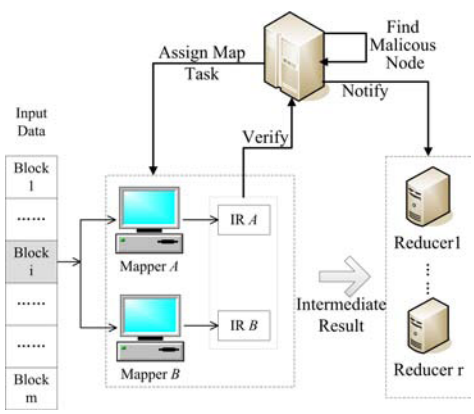


Fig. 2 Collusion-resistant mapper verification.

To find the periodical attackers, the master records the verification relationship of the two mappers that participate in the verification and analyzes this information after a certain period, during which k times of replication verification are done. Confirmed malicious workers are no longer scheduled. When a malicious worker provides a false result, one of two situations could occur. First, if the other replica is processed by a worker from the same collusive group, the false result of the two nodes passes verification and slips into the next phase. Second, if the other replica is processed by a non-collusive worker, the results fail verification, and the master has to reschedule two workers to perform computing. Failure leads to a waste of computing resources. By identifying and removing malicious workers as early as possible, rescheduling of computing resources can be prevented and the accuracy of computing results can be improved.

3.2 Identifying Malicious Workers

To verify if a worker can be trusted, the consistency of the actions of all workers participating in the computing must be examined. Benign workers always offer the same results as do malicious ones from the same collusive group. By modifying the integrity attestation graph-based method proposed in [11] and exploiting the periodic attack model, we give the definition of *integrity attestation graph* for mappers in an open MapReduce computing environment.

Definition 1: (**Integrity attestation graph (IAG)**) a weighted undirected graph utilized to express the consistency of results between mappers. The nodes in IAG represent the workers in map computing, and the edges denote the consistency between two nodes. Edges have a value of 0 or 1, and the two nodes containing an edge form a *verification couple*. If an edge between two nodes has a value of 0, it means that the nodes have provided different results. Then the two nodes form an *inconsistency pair*. When the edge between two nodes has a value of 1, it means the two nodes have not provided different results yet and are thus called a *temporary consistency pair*.

Two nodes identified as inconsistent have a determined inconsistent relationship forever, no matter what result will be in the subsequent verification between them. However, if two nodes are in a temporary consistent relationship, three cases are possible. First, the two nodes have not been verified as a pair. Second, they are benign nodes or malicious ones from the same collusive group; thus, they act in the same way. Third, the two nodes belong to different groups but the malicious node did not misbehave in the previous verification between this pair. In the second case, the edge between the two nodes is always 1. In the first and third cases, additional verification tests can expose the inconsis-

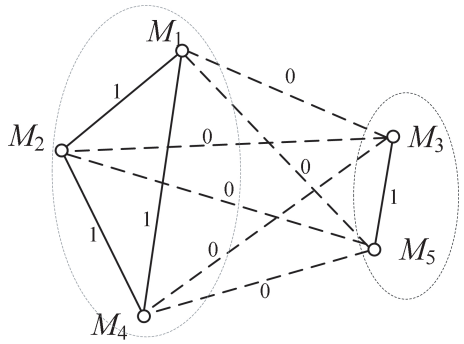


Fig. 3 Integrity attestation graph.

tency between nodes.

Figure 3 shows an IGA containing five mappers. When two nodes $\langle M_i, M_j \rangle$ complete the same task and the two results match each other, the edge between the two nodes is 1; otherwise, the edge is 0. The temporary consistency pairs in Fig. 3 are $\langle M_1, M_2 \rangle$, $\langle M_2, M_4 \rangle$, $\langle M_1, M_4 \rangle$, and $\langle M_3, M_5 \rangle$. The other pairs are inconsistency pairs.

The verification couples composed of a benign node and a malicious one finally form an inconsistency relationship after a certain number of verifications because the malicious node misbehaves with a certain probability. We utilize the *consistency clique* to analyze the consistency relationship between nodes.

Definition 2: (Consistency clique) the maximum completed IAG sub-graph which has at least two nodes and the value of every edge is 1. Nodes in the same consistency clique provide the same results.

As shown in Fig. 3, after sufficient verification, IAG indicates that M_1, M_2, and M_4 form a consistency clique, whereas M_3 and M_5 form another one.

Given that benign nodes always provide the same results, there is at least one consistency clique containing all benign nodes [11]. Based on experience, we assume that the number of benign nodes is larger than that of malicious nodes. Thus, if a node does not belong to any maximum clique with a number of nodes larger than $\lfloor N/2 \rfloor$, the node is definitely malicious [11]. So we transform the problem of identifying malicious nodes into MCP based on IAG. MCP is a classic NP complete problem that can be solved by many algorithms. We employ the *Bron-Kerbosch* (BK) clique-finding algorithm [12], [13] as an example.

In our algorithm, k pairs of nodes are first verified; IAG is then modified accordingly. For a verification couple $\langle V_i, V_j \rangle$, if edge $E(i, j)$ in IAG is 0, the pair is already inconsistent and the edge value requires no alteration. However, if $E(i, j)$ is 1 (i.e., the pair is temporarily consistent) and verification nodes i and j provide different results this time, $E(i, j)$ is altered to 0 to illustrate that the two nodes are identified as inconsistent. IAG can then be simplified, and the maximum clique analysis algorithm *BK_MaxClique* is adopted to obtain all maximum cliques in IAG. The candidate mapper set is then traversed. A node that does not belong to any maximum clique with the number of nodes larger than $\lfloor N/2 \rfloor$ is removed from the candidate set. The algorithm of *malicious nodes identifying* through IAG is shown as Algorithm 1.

Algorithm 1: *malicious nodes identifying*

Input:
IAG of the system, $G = \langle V, E \rangle$;
The verification relationship R of k pairs;
Candidate nodes set, PN;
Malicious nodes set, MN;

Procedure:
1: **For** each pairs $\langle V_i, V_j \rangle$ in R, **do**:
2: **If** $E(i, j) == 1$ && $R\langle i, j \rangle == 0$
3: **Then**
4: $E(i, j) = 0$;
5: Simplify G, delete all the nodes whose edges to all other nodes are 0, and generate G' that used for maximum clique analysis;
6: Call *BK_MaxClique*(G'), and get all maximum cliques contained in G', MC_i;
7: Find all cliques with number of nodes larger than $\lfloor N/2 \rfloor$, MC_b;
8: **For** each node N_i in PN, **do**:
9: **If** N_i does not belong to any MC_b
10: **Then**
11: $PN = PN - \{N_i\}$;
12: $MN = MN + \{N_i\}$;

When the system begins to operate, PN contains all nodes in V and MN is Φ. All edges in IAG have a value of 1, indicating that all nodes are temporarily consistent by default. After every k times of replication verification, the algorithm is called once. The algorithm checks if malicious nodes exist and removes them from the candidate set.

3.3 Verification Couple Selection

If two workers provide the same results during verification, the workers are only identified as a temporary consistent pair; inconsistency may be exposed in subsequent tests. When the two workers provide different results, they are determinately inconsistent. This result is unchangeable even if additional verification is performed. Therefore, the inconsistency relationship in IAG must be determined as soon as possible to quickly identify malicious mappers.

When a node is identified as malicious, the edges between this node and each of the others can be analyzed. Assuming that the detected malicious node is A_i, if node B forms an edge with A_i (with a value of 1), then the following instances could occur: (1) A_i and B belong to the same collusive group; (2) A_i and B belong to different collusive groups but both did not misbehave; and (3) B is a benign node, and A_i did not misbehave. If another node C forms an edge with A_i (with a value of 0), then C must not belong to the same collusive group as A_i. We define these two kinds of nodes as follows.

Definition 3: Assuming that malicious nodes $A_1, \ldots, A_s$ are found in IAG $G = \langle V, E \rangle$, then for every A_i ($1 \leq i \leq$

s), the other nodes can be divided into two groups (*collusive set S1* and *opposed set S0*) based on the edge values from these nodes to A_i. *S1* contains all the nodes that form an edge with A_i (with a value of 1); that is, $S1 = \{b \mid b \in V, \text{ and } E(A_i, b) = 1\}$. Nodes that form an edge with A_i (with a value of 0) form set *S0*. $S0 = \{c \mid c \in V, \text{ and } E(A_i, c) = 0\}$.

Theorem 1: For a malicious node A_i, the nodes in its corresponding collusive group must also belong to $S1(A_i)$. The nodes in $S0(A_i)$ do not belong to the collusive group of A_i.

Proof: If node B belongs to the collusive group of A_i and $E(A_i, B) = 0$, then it means A_i has provided different results as that of B in a certain test. This contradicts the assumption that the nodes are in the same group. Thus, the value of $E(A_i, B)$ has to be 1; that is, B belongs to $S1(A_i)$. Similarly, we can prove that the nodes in $S0(A_i)$ are not contained in A_i's group.

□

The nodes in $S1(A_i)$ may be the accomplices of A_i; however, the nodes in $S0(A_i)$ must not be accomplices. Thus, we select nodes from these two different groups to quickly identify inconsistency among nodes. Based on A_i's information, we therefore select one node from the $S0(A_i)$ and select the other from $S1(A_i)$ to increase the probability of inconsistency detection.

When s malicious nodes have been identified in the system, each of them contains certain information. We must determine how many pairs corresponding to a malicious node are required for verification when total k verification couples are assigned. We define the malicious influence of a node below.

Definition 4: (**Malicious Influence**) The malicious influence of a node depends on the scale of the collusive group the node belongs to. The larger the scale is, the greater the influence is. Assumed that there are m malicious node totally, the malicious influence of malicious node A_i can be evaluated as:

$$MInfl(A_i) = \frac{|S1(A_i)|}{\sum_{j=1}^{m} |S1(A_j)|} \tag{1}$$

Theorem 2: Given that IAG $G = \langle V, E \rangle$, where V contains n nodes, including p benign nodes and $n - p$ malicious nodes, and the malicious nodes form s collusive groups. The misbehavior probability of one malicious node is b $(0 < b \leq 1)$, which is independent to other nodes and same for all collusive groups. Then after a complete test (each node is paired and tested with all other nodes in the graph), the higher the expectation of *MInfl* of one malicious node is, the greater the scale of its collusive group would be.

Proof: After a complete test (each malicious node A_i has been paired and tested with all other nodes in the system), A_i's *S1* is composed of three kinds of nodes: (1) malicious nodes in its collusive group, and the number of these nodes is denoted by Q_i; (2) benign nodes, and during the couple-verification with A_i, A_i did not misbehave. The number of these nodes is denoted by X; and (3) malicious nodes from other collusive groups, and during the couple-verification with A_i, both the node and A_i did not misbehave. The number of these nodes is denoted by Y. Here we assume that the probability that non-collusive malicious workers return identical wrong results for a specific task is 0.

Thus, the number of nodes in *S1* of A_i is:

$$|S1(A_i)| = Q_i + X + Y \tag{2}$$

Where X is a binomial random variable with parameter $(p, 1 - b)$, and Y is a binomial random variable with parameter $(n - p - Q_i, (1 - b)^2)$. Then the expectation of $|S1(A_i)|$ is:

$$E(|S1(A_i)|) = Q_i + p * (1 - b) + (n - p - Q_i) * (1 - b)^2$$

Therefore, we have derived the Q_i:

$$Q_i = \frac{E(|S1(A_i)|) - p * (1 - b) - (n - p) * (1 - b)^2}{1 - (1 - b)^2}$$

We assume that A_i and A_j belong to collusive groups i and j, respectively, and $E(MInfl(A_i)) \geq E(MInfl(A_j))$. According to the definition of *MInfl*, we have $E(|S1(A_i)|) \geq E(|S1(A_j)|)$. So,

$$Q_i - Q_j = \frac{E(|S1(A_i)|) - E(|S1(A_i)|)}{1 - (1 - b)^2} \geq 0$$

Therefore, we obtain $Q_i \geq Q_j$.

□

From Eq. (2), we find that besides the number of A_i's collusive nodes, $S1(A_i)$ can be affected by several random factors, such as the random behaviors of A_i and other non-collusive malicious nodes when doing the pair-wise verification. So in terms of statistics, the decision based on the result of a single run of experiment is not deterministic. In this situation, generally, the mean value of the results of multiple repeated experiments is used to evaluate the effectiveness of the indicator. In Theorem 2, we have proved that the scale of a malicious node's collusive group would be greater when its expectation of *MInfl* is higher, so we have reasons to believe that the $MInfl(A_i)$ is a good indicator of the scale of a malicious node's collusive group.

If totally k verification couples are required for verification, we allocate different quotas of couples according to the malicious influence of every malicious node. More verification couples are allocated for the nodes with greater influence. Considering malicious influence, $MInfl(A_i) * k$ verification couples are selected corresponding to malicious node A_i.

We describe the algorithm of *verification couples selection* as Algorithm 2. In the algorithm, we don't use the IAG-instructed scheduling (line 2–13) at all times. First, at the beginning of execution, there is no malicious node identified in the system, so the algorithm above will randomly

Algorithm 2: *verification couples selection*

1:**If** the set of malicious nodes is not empty, and the number of newly identified malicious nodes by previous call of *Algorithm 1* is less than ε, **Then**
2: {
3: **For** each known malicious node A_i **do**:
4: evaluate $S1(A_i)$ and $S0(A_i)$;
5: **For** each known malicious node A_i **do**:
6: compute A_i's $MInfl(A_i)$;
7: According to each A_i's $MInfl(A_i)$, compute the number of verification couples corresponding to A_i, i.e. $MInfl(A_i) * k$
8: Traverse the set of identified malicious nodes, and for every node, **do**:
9: **For** $i = 1$ to $MInfl(A_i) * k$ **do**
10: Randomly choose node x in $S1(A_i) \bigcap PN$;
11: Randomly choose node y in $S0(A_i) \bigcap PN$;
12: Define $\langle x, y \rangle$ as a verification pair.
13: }
14: **Else**
15: Select k verification couples in PN randomly

select verification couples. Second, after sufficient couple tests, almost all malicious nodes are found and the *S1* set of a malicious node is primarily composed of its collusive nodes, which are identified and excluded from the scheduling. If we only use the IAG-instructed scheduling, finally there will be few nodes to select in *S1*. Thus, selecting couples randomly is more appropriate in this case. Here we use ε, which is usually not bigger than 1, as the switch of two selections. For example, if the value of ε takes 0.5, it means that the random selecting will be used when there is no malicious node newly detected in the recent two callings of Algorithm 1. Generally, we make ε equal to 1, that is, if no malicious node is newly identified in the recent calling of Algorithm 1, the random selecting will be used.

Algorithm 2 facilitates the rapid identification of malicious nodes by determining inconsistencies between allocated replicas; however, the algorithm generates more computing failures in the early stage of execution. As more malicious nodes are found and removed, the efficiency of the computation and the accuracy of the results are improved significantly because the proportion of benign nodes is high.

4. Analysis and Evaluation

The performance of the proposed scheme is evaluated analytically and experimentally in this section. Analysis shows that our scheme can effectively identify malicious workers in collusive and non-collusive attacks. Compared with the methods adopted in other studies, our method improves result accuracy and reduces overhead of verification. And our method avoids centralized verification, which will cause a bottleneck in performance.

4.1 Theoretical Analysis

The main performance indices of the computation environment with the verification scheme are analyzed in this section to verify the effectiveness of the proposed method. The efficiency in discovering inconsistency is compared between the methods of our verification couple selection and the random verification couple selection.

We evaluate the result verification mechanism of mappers based on two important performance indices: *accuracy* and *overhead*. The accuracy (AC) of a map task is the probability that a task would provide a good result to the master; the master would then release the result to the reducer. The overhead (OH) of a map task is the average number of executions launched by the worker for each task. AC is employed to evaluate the quality of the result released to the reduce phase, and OH is employed to evaluate the efficiency of computation. A good verification mechanism has high result accuracy and low computation overhead.

Malicious nodes are identified and removed from the mapper scheduling in our scheme. Performance is enhanced by reducing the ratio of malicious workers. AC and OH at a fixed ratio of malicious workers are analyzed, and the variation trend of these two indices at a reduced ratio of malicious workers is presented.

Our analysis model assumes a cloud environment containing a large number of mappers, so the number of nodes is so large that we can ignore the difference between with or without replacement of mappers. It is assumed that there are N mappers in the environment, M of which are malicious. We define the ratio of malicious workers as $r = M/N$. M malicious workers include both collusive and non-collusive workers. For Simplicity, we define c as the portion of collusive workers in M malicious workers. The misbehavior probability of one malicious node is independent to others and same for all collusive groups. The misbehavior probability is defined as b. Two collusive workers assigned the same task misbehave at the probability of b. We assume that malicious workers always know when they are assigned the same task. For simplicity, here we assume the verification couple nodes are randomly chosen.

We focus on collusive attacks in the discussion of the accuracy of a map task. Since that each instance of misbehavior in the non-collusion strategy will be detected, the accuracy of this case is always 1. In the collusive mode, the same false result provided by two malicious workers will pass the verification and is released to the reducer, thereby affecting accuracy.

Accuracy is influenced under two cases:

a. When a task is assigned to two workers in the same malicious group, the false result passes verification. The probability of this case is r^2c^2b.

b. When the master detects an inconsistency, two other replicas are scheduled; thus, accuracy relies on the accuracy of the new schedule. Rescheduling occurs when (1) the verification couple of the schedule includes one benign worker

and one malicious worker, and the malicious worker misbehaves; or (2) the verification couple includes two non-collusive malicious workers, at least one of which misbehaves. Thus, the probability of rescheduling (RP) is

$$RP = 2(1-r)rb + 2r^2(1-c)(1-(1-b)^2) \quad (3)$$

We obtain the cheat probability (CP) of a map task by combining the two cases.

$$CP = r^2 \cdot c^2 \cdot b + RP \cdot CP$$

Therefore,

$$CP = \frac{r^2c^2b}{1-2(1-r)rb-2r^2(1-c)(1-(1-b)^2)}$$

Given that $AC = 1 - CP$, we obtain

$$AC = 1 - \frac{r^2c^2b}{1-2(1-r)rb-2r^2(1-c)(1-(1-b)^2)} \quad (4)$$

OH can be calculated with the same principle. In our model, rescheduling occurs when the master discovers the inconsistency. The probability of rescheduling is equal to (3). The OH of rescheduling is 2+OH. Otherwise, OH is 2 because rescheduling is unnecessary. Thus, adding these OH values yields

$$OH = RP \cdot (2 + OH) + (1 - RP) \cdot 2$$

Therefore, we derive OH as

$$OH = \frac{2}{1-2(1-r)rb-2r^2(1-c)(1-(1-b)^2)} \quad (5)$$

Figure 4 shows the simulated relationship between the ratio of malicious workers and AC based on (4) when b is 0.5, while c gets 0.5 and 1.0. Reducing the ratio of malicious worker (r) significantly improves accuracy. When c is equal to 1, accuracy increases by 13% as r is reduced from 0.45 to 0.10. Figure 5 shows the simulated relationship between r and OH based on (5) when b is 0.5 and c gets 0.5 and 1.0. Reducing r significantly reduces overhead. Overhead drops from 3.3 to 2.2 with the reduction of r from 0.45 to 0.1.

We compare the proposed verification couple selection method's efficiency in discovering inconsistency with that of random couple selection. If we select two replicas randomly under the same assumption as above, the probability of inconsistency is equal to (3). In our method, the other nodes are divided into two sets ($S1(A_i)$ and $S0(A_i)$) based on the verification relationship of the nodes with a certain malicious node A_i. For simplicity, following discussion is based on the assumption that each couple among the mappers in the environment has already been verified at least once. There are three cases when a node belongs to $S1$: (1) the node is in the same collusive group as A_i; (2) the node is a benign node, and A_i does not misbehave during couple verification with this node; (3) the node is a malicious node and non-collusive with A_i, and both the node and A_i do not misbehave during couple verification with these two nodes. The nodes in $S0$ are either of the following two cases: (1) the node is a benign worker, and A_i misbehaves during couple verification with this node; (2) the node is a malicious node and non-collusive with A_i, and at least one of the nodes and A_i misbehave during couple verification with these two nodes. Thus, the expectation of the numbers of the elements of $S1(A_i)$ and $S0(A_i)$ are

$$E(|S1|) = nrc - 1 + n(1-r)(1-b) + nr(1-c)(1-b)^2$$

$$E(|S0|) = n(1-r)b + nr(1-c)b$$

In the proposed method of verification couple selection, an inconsistency pair can be determined in the following three procedures:

(1) The node selected in $S1$ is in the same collusive group as A_i, and this node misbehaves during couple verification of this time.

(2) The node selected in $S1$ is benign, whereas the node selected in $S0$ is malicious and misbehaves during couple verification of this time.

(3) The node selected in $S1$ is non-collusive with A_i and misbehaves during couple verification of this time.

Combining all the above cases, the probability of inconsistency discovery of our method is

$$InC\_DR_{ours} = \left(\frac{nrc-1}{E(|S1|)} + \frac{n(1-r)(1-b)}{E(|S1|)} * \frac{nr(1-c)b}{E(|S0|)} + \frac{nr(1-c)(1-b)^2}{E(|S1|)}\right) * b$$

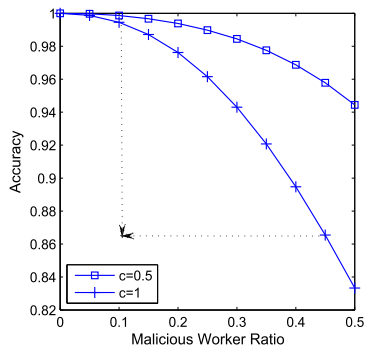


Fig. 4 Accuracy vs malicious worker ratio.

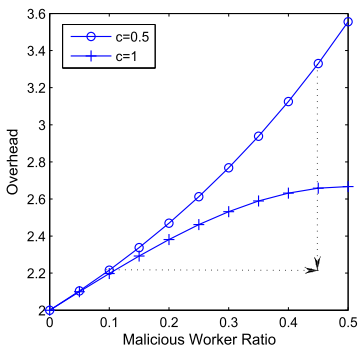


Fig. 5 Overhead vs malicious worker ratio.

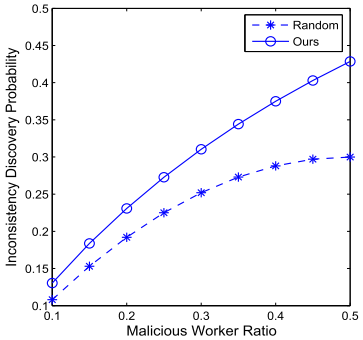


Fig. 6 Comparison of inconsistency discovery probability.

Our analysis model assumes a cloud environment that contains a large number of workers, specially, mappers. So due to the large value of n, here we can simplify the equation as:

$$InC\_DR_{ours} = \left(\frac{rc}{\Delta_1} + \frac{(1-r)(1-b)}{\Delta_1} * \frac{r(1-c)b}{\Delta_2} + \frac{r(1-c)(1-b)^2}{\Delta_1}\right) * b \quad (6)$$

where:

$$\Delta_1 = rc + (1-r)(1-b) + r(1-c)(1-b)^2$$

$$\Delta_2 = (1-rc) * b$$

The probability of inconsistency discovery of random selection is

$$InC\_DR_{random} = 2r(1-r)b + 2r^2(1-c)(1-(1-b)^2) \quad (7)$$

Figure 6 shows the simulated probabilities of inconsistency discovery based on (6) and (7) when r has different values. We let $c = 1$ and $b = 0.6$. It is showed that our method achieves higher probability.

4.2 Experimental Evaluation

Firstly, to test the effectiveness of our proposed scheme in a large scale cluster, we deployed the simulation experiment environment with 100 mapper nodes. Three performance indices were tested in the simulation experiments: (1) the *detection rate* of malicious workers and the (2) *accuracy* and (3) *overhead* of a map task. Detection rate is the portion of identified malicious worker out of the total malicious workers. It is utilized to evaluate the efficiency of malicious worker discovery based on IAG.

We tested detection rate both in collusive and non-collusive attack strategies, with different values of misbehaving probability b and malicious worker ratio r. For simplicity, we assume that in a collusive attack mode, only one collusive group exists ($c = 1$), while in a non-collusive attack, all malicious workers are not collusive ($c = 0$). The algorithm of malicious nodes identifying is employed after every 10 couples' verification. Figures 7 and 8 show the relations between the number of verification couples and the detection rate of malicious workers, both in collusive and non-collusive attacks, when r and b take different values. It is showed that our method has the same effectiveness in detecting malicious workers on both collusive and non-collusive attack modes. In Fig. 7, r is fixed at 0.3, and the detection rate becomes higher when b increases, because the malicious worker is easier to misbehave and to be discovered. In Fig. 8, the value of b is fixed at 0.5, and the figures show that the higher r is, the better the detection rate is, because more malicious workers are easier to form the clique and to be detected.

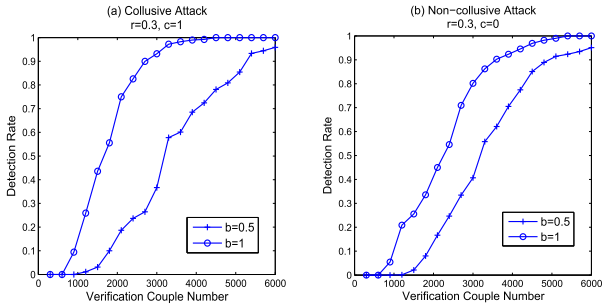


Fig. 7 Relations between number of verification couples and detection rate of malicious workers (with different b).

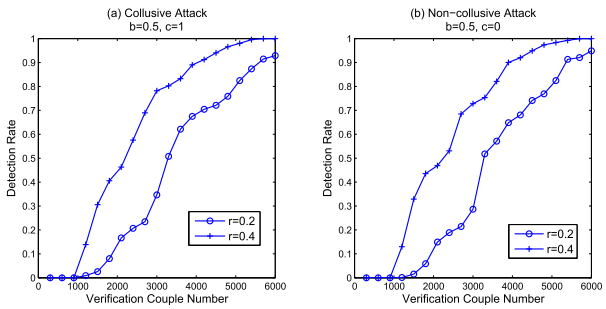


Fig. 8 Relations between number of verification couples and detection rate of malicious workers (with different r).

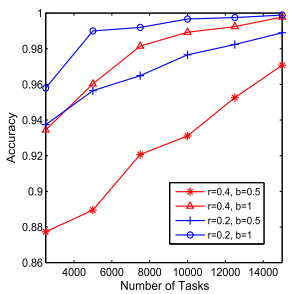


Fig. 9 Accuracy vs number of map tasks.

Next, we tested the influence on AC and OH when the total number of map tasks varied. Figure 9 shows AC under different amounts of map tasks when malicious worker ratio r and probability of misbehaving b have different values. In general, the larger the amount of tasks is, the higher AC is. Given that almost all malicious workers are identified and removed in the earlier stage of execution, the succeeding tasks are executed only by benign workers. Thus, the mean accuracy of the entire task rises when more tasks

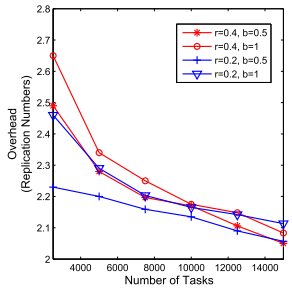


Fig. 10 Overhead vs number of map tasks.

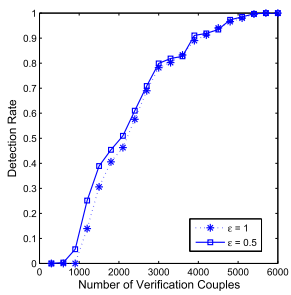


Fig. 11 Comparison of detection rate with different ε.

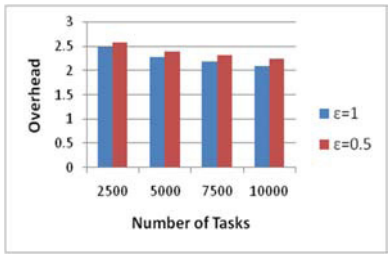


Fig. 12 Comparison of overhead with different ε.

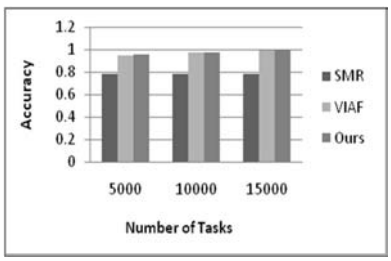


Fig. 13 Comparison of the accuracy.

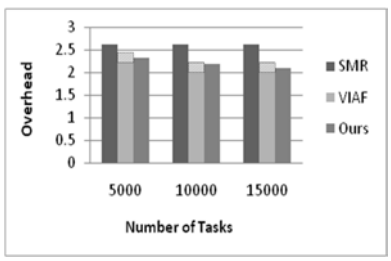


Fig. 14 Comparison of the overhead.

are involved. For a fixed number of tasks, an increase in r reduces accuracy owing to more collusion happening; an increase in b improves accuracy because the malicious workers are easily exposed. Figure 10 shows OH under different numbers of map tasks. OH is the mean number of replications required to compute a map task. Similarly, OH decreases when the total number of tasks is increased. For a fixed number of tasks, an increase in r increases OH because identifying all malicious workers becomes more difficult; an increase in b increases OH because rescheduling is easier to occur. After approximately 3000 tasks done, our scheme discovers all malicious workers, hence the succeeding task is computed only by benign workers. Then we achieve high performance where AC is nearly 100% and OH is close to the ideal value of 2.

Next, we tested the influence on the performance when ε takes different values in Algorithm 2. We take the value of ε as 1 and 0.5 respectively. In Fig. 11, it is showed that with smaller numbers of verification couples, the detection rate with lower ε is higher, for Algorithm 2 is called more times. With increasing number of couple tests, the difference of detection rate is not apparent. Since there are more times of rescheduling when ε takes a lower value, the overhead is higher when ε takes value of 0.5, as shown in Fig. 12.

For comparison, we also tested the accuracy and overhead in the simulated SecureMR and VIAF verifications. The master randomly selects verification couples in SecureMR; in VIAF, a trustworthy role called verifier is introduced to conduct a sampling test on the same result. All the three schemes are based on the assumption that the number of benign nodes is larger than that of malicious nodes. In the experiments, the total number of tasks takes value of 5000, 10000, and 15000. The malicious ratio r is 0.4, and the probability of misbehaving b is fixed at 1. The system has only one collusive group ($c = 1$). The verification probability of tasks in VIAF is 20%, while it is 100% in other two schemes. The results are showed in Figs. 13 and 14. Since there is no attestation on malicious nodes in SecureMR, its AC is the lowest and the OH is the highest. In Fig. 13, the three results of AC of our proposed scheme are a bit higher than those of VIAF. In Fig. 14, although OH of mappers in VIAF is lower than ours, another overhead (>20%) induced by the centralized the verifier exists in VIAF. In the experiment of VIAF, we set the quiz threshold to 1 and only 20% tasks were verified. So if we use more quiz tests or set verification probability higher, its AC will be higher than ours. But consequently, the centralized overhead of VIAF will also increase a lot in those cases, which may cause a bottleneck in performance.

Secondly, to evaluate the overhead in terms of execution time, we create a prototype of our mapper scheduling mechanism with Hadoop 0.20.2 [14]. We deploy 12 machines to construct a MapReduce environment; one is running as both the master and worker, the others are running as workers. All hosts has similar configurations of hardware and software, which is with Intel(R) Core(TM)2 DUO CPU 2.66 GHz, 512MB of memory and 20GB disk, and CentOS release 5.5 and Sun JDK 6. The experiments are conducted

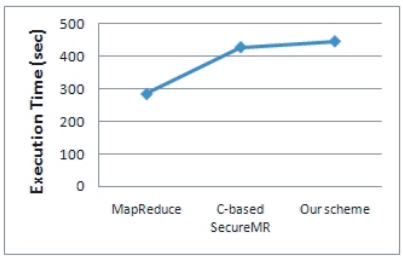


Fig. 15 Comparison of execution time.

by using Hadoop WordCount application. The complete job requires 60 map tasks and 25 reduce tasks. The data size is 1G. The number of malicious mappers is 4 and the misbehaving probabilities of malicious nodes are fixed at 1. We implement both algorithms of *malicious nodes identifying* and *verification couples selection* mentioned in the previous section. The master utilizes the algorithm of verification couples selection to select the verification couples in the task assignment. After every 20 couples' verification, the algorithm of malicious nodes identifying is employed to search for malicious workers.

The results are showed in Fig. 15. We compared the execution time with the naive MapReduce and the Commitment-based SecureMR. Comparing with the naive MapReduce, the performance overhead caused by our verification mechanism is about 56%. And the execution time in our scheme is higher than that in the Commitment-based SecureMR (about 4%). The extra overhead of our scheme includes: (1) Duplication of computation tasks, (2) the delay of task launching in order to defend strategic attackers, (3) the re-computation overhead of tasks that don't passing the duplication test, and (4) the overhead caused by the malicious nodes identifying and by the selection and execution of k verification couples. As to the Commitment-based SecureMR, the first and second factors of overhead are same to ours, and there is no extra overhead of malicious nodes identification and the selection of verification couples. But rescheduling in that scheme happens more times than ours, which brings more overhead, while the overhead of rescheduling in our scheme will be gradually reduced with the increasing number of tasks.

5. Related Works

The result verification of outsourcing is an emergent topic along with distributed computing modes. Replication and voting features [15] for redundant computing are applied such that multiple computing nodes will perform the same job; the result is accepted when it is submitted by more than half of the total nodes. Sampling techniques address the resource cost of replication, involving result-based sampling [16] and test job injection sampling [17]. With sampling techniques, computation results are verified and trusted with a certain probability. Checkpointing deals with result verification for sequential computation [18].

Currently, research studies concerning result verification for mass data processing of MapReduce focus on the computation integrity of different levels. Considering the untrustable SPs, Chu Huang et al. proposed a watermark injection method to verify if the computation is completed correctly [19]. For the untrustable participating nodes in an open MapReduce environment, Wei Wei et al. proposed an integrity protection mechanism called SecureMR, which uses two-copy replication to verify the result in the map phase [9]. Yongzhi Wang and his colleagues introduced the verifier role in the MapReduce computing model [10], which samples and re-computes the results passed the replication verification, to defend collusion attack. Z. Xiao et al. used a set of trusted auditing nodes to record the results generated by various phases of MapReduce [20]. The cheating nodes can be located by re-computing the results.

Different from previous work, our research focuses on the identification of malicious mappers in both collusive and non-collusive attack strategies without introducing the centralized re-computing verification.

The IAG is first used in [11] to detect the malicious components in the data streaming process. But the research focuses on finding the different potential attack patterns. In our research, the IAG-based method is used to detect malicious mappers in the open MapReduce system, and the definition of IAG is modified according to our proposed attack model. Furthermore, in order to accelerate the identification rate, we proposed some heuristics to schedule replication verification pairs.

6. Conclusion

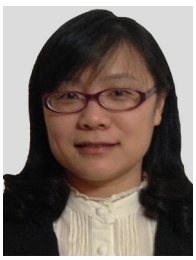
In this paper, we have presented the design of VAWS, a trusted worker scheduling framework of MapReduce over open system. VAWS detects collusive attackers and assures the integrity of data processing without extra centralized re-computation. We propose a method of identifying malicious mapper based on IAG, and design the verification-couple selection method based on the influence of malicious workers with IAG guidance. Theoretical analysis and experiment results show that our method can effectively detect malicious workers under both collusive and non-collusive attacks. Compared with other related methods, our method achieves higher accuracy while imposing only a low overhead computation. For this study is based on the assumption that reducers are trusted, we will focus on guaranteeing integrity without this assumption in future research.

Acknowledgments

This work was supported by the National Basic Research Program of China under Grant No. 2011CB302600, the National Natural Science Foundation of China under Grant No. 61161160565, the HGJ Major Project of China under Grant No. 2013ZX01040-002, NSFC No. 61379146 and No. 61272483, NSTPP No. 2012BAH13F04 and the Fund No. KJ-13-105.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Proc. Conference on Symposium on Operating Systems Design & Implementation, p.10, San Francisco, CA, USA, Dec. 2004.
[2] Running Hadoop on Amazon EC2, http://wiki.apache.org/hadoop/AmazonEC2
[3] F. Costa, L. Veiga, and P. Ferreira, "Boinc-Mr: Mapreduce in a volunteer environment," Proc. Conference on the Move to Meaningful Internet Systems: OTM, pp.425–432, Rome, Italy, Sept. 2012.
[4] L. Heshan, M. Xiaosong, A. Jeremy, et al., "Moon: Mapreduce on opportunistic environments," Proc. 19th ACM International Symposium on High Performance Distributed Computing, pp.95–106, Chicago, Illinois, USA, June 2010.
[5] F. Marozzo, D. Talia, and P. Trunfio, "A framework for managing Mapreduce applications in dynamic distributed environments," Proc. 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp.149–158, Ayia Napa, Cyprus, Feb. 2011.
[6] K. Lee, T.W. Choi, A. Ganguly, D.I. Wolinsky, P.O. Boykin, and R. Figueiredo, "Parallel processing framework on a P2P system using map and reduce primitives," Proc. 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), pp.1602–1609, Alaska, USA, May 2011.
[7] F. Marozzo, D. Talia, and P. Trunfio, "P2P-MapReduce: Parallel data processing in dynamic cloud environments," J. Computer and System Sciences, vol.78, no.5, pp.1382–1402, May 2012
[8] E. Marinelli, "Hyrax: Cloud computing on mobile devices using Mapreduce," CMU, Report No.: CMU-CS-09-164, 2009.
[9] W. Wei, J. Du, T. Yu, and X. Gu, "SecureMR: A service integrity assurance framework for mapreduce," Proc. 25th Annual Computer Security Applications Conference (ACSAC 09), pp.73–82, Honolulu, Hawaii, USA, Dec. 2009.
[10] Y.Z. Wang and J.P. Wei, "VIAF: Verification-based integrity assurance framework for MapReduce," Proc. IEEE International Conference on Cloud Computing (Cloud 11), pp.300–307, Washington DC, USA, July 2011.
[11] J. Du, W. Wei, X. Gu, and T. Yu, "Runtest: Assuring integrity of dataflow processing in cloud computing infrastructures," Proc. 5th ACM Symposium on Information, Computer and Communications Security, pp.293–304, Beijing, China, April 2010.
[12] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," Commun. ACM, vol.16, no.9, pp.575–577, Sept. 1973.
[13] F. Cazals and C. Karande, "Note: A note on the problem of reporting maximal cliques," Theor. Comput. Sci., vol.407, no.1-3, pp.564–568, Nov. 2008.
[14] Hadoop, "Apache Hadoop," [Online]. http://hadoop.apache.org
[15] M. Taufer, D. Anderson, P. Cicotti, and C. Brooks III, "Homogeneous redundancy: A technique to ensure integrity of molecular simulation results using public computing," Proc. 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 05), Workshop 1, p.119a, Denver, Colorado, USA, April 2005.
[16] W. Du, J. Jia, M. Mangal, and M. Murugesan, "Uncheatable grid computing," Proc. 24th International Conference on Distributed Computing Systems (ICDCS 04), pp.4–11, Tokyo, Japan, March 2004.
[17] Z. Shanyu and V. Lo, "Result verification and trust-based scheduling in Peer-to-Peer grids," Proc. 5th IEEE International Conference on Peer-to-Peer Computing (P2P 05), pp.31–38, Konstanz, Germany, Aug. 2005.
[18] F. Monrose, P. Wycko, and A.D. Rubin, "Distributed execution with remote audit," Proc. Network and Distributed System Security Symposium (NDSS 99), pp.103–113, San Diego, California, USA, Feb. 1999.
[19] C. Huang, S.C. Zhu, and D.H. Wu, "Towards trusted services: Result verification schemes for MapReduce," Proc. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 12), pp.41–48, Ottawa, Canada, May 2012.
[20] Z.F. Xiao and Y. Xiao, "Accountable MapReduce in cloud computing," Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS 11), pp.1082–1087, Shanghai, China, April 2011.



Yan Ding received the B.E. and M.E. degrees in Computer Science from National University of Defense Technology (NUDT), China in 2000 and 2002, respectively. She is now an instructor in School of Computer Science, NUDT. Her research interests include information security, operating system and cloud computing.

Huaimin Wang received his Ph.D. in Computer Science from National University of Defense Technology (NUDT) in 1992. He is now a professor and chief engineer in department of educational affairs, NUDT. He has been awarded the "Chang Jiang Scholars Program" professor by Ministry of Education of China, and the Distinct Young Scholar by the National Natural Science Foundation of China. His current research interests include middleware, software Agent, trustworthy computing.

Lifeng Wei received his Ph.D. degree in Computer Science from National University of Defense Technology (NUDT), China in 2002. He is now an associate professor in the School of Computer Science, NUDT. His research area is system software and information security.

Songzheng Chen received his B.S., M.S. in Computer Science from National University of Defence Technology (NUDT), China, in 1993 and 1999, respectively. He is now an associate professor in the School of Computer, NUDT. His research interests include system security and trustworthy computing.

Hongyi Fu received his B.S., M.S. and Ph.D. in Computer Science from National University of Defense Technology, China in 2000, 2004 and 2011, respectively. His Ph.D. thesis focuses on the fault tolerance techniques of OpenMP programming. His research interests include high performance computing, parallel computing, parallel compiling and fault tolerance.

Xinhai Xu received his B.S., M.S. and Ph.D. degrees in Computer Science from the National University of Defense Technology (NUDT), China in 2006, 2008 and 2012, respectively. He is now an assistant professor in the School of Computer Science, NUDT. His research interests include high-performance computing architectures and performance evaluation.