Future Generation Computer Systems 55 (2016) 87–100

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Toward integrity assurance of outsourced computing — a game theoretic perspective

Yongzhi Wang [a,b,*], Jinpeng Wei [c], Shaolei Ren [d], Yulong Shen [a]

[a] School of Computer Science and Technology, Xidian University, 2 South Taibai Road, Xi'an, Shaanxi, PR China
[b] Florida International University, ECS 254 11020 SW 8th Street, FL, USA
[c] Florida International University, ECS 389 11020 SW 8th Street, FL, USA
[d] University of California, Riverside, CA, USA

HIGHLIGHTS

- Our method can ensure high result integrity in outsourced computing systems.
- Our algorithm can guarantee the highest result integrity under system restrictions.
- We proved the correctness of the proposed algorithms.
- We performed experiments to show the effectiveness of the proposed algorithms.

ARTICLE INFO

Article history:
Received 25 March 2015
Received in revised form
12 July 2015
Accepted 19 August 2015
Available online 3 September 2015

Keywords:
Outsourced computing
Game theory
Integrity assurance
Task scheduling

ABSTRACT

Outsourced computing is gaining popularity in recent years. However, due to the existence of malicious workers in the open outsourced environment, offering high accuracy computing services is critical and challenging. A practical solution for this class of problems is to replicate outsourced tasks and compare the replicated task results, or to verify task results by the outsourcer herself. However, since most outsourced computing services are not free, the portion of tasks to be replicated or verified is restricted by the outsourcer's budget. In this paper, we propose *Integrity Assurance Outsourced Computing* (IAOC) system, which employs probabilistic task replication, probabilistic task verification and credit management techniques to offer a high accuracy guarantee for the generalized outsourced computing jobs. Based on IAOC system, we perform theoretical analysis and model the behaviors of IAOC system and the attacker as a two-player zero sum game. We propose two algorithms, *Interactive Gradient Descent* (IGD) algorithm and *Tiered Interactive Gradient Descent* (TIGD) algorithm that can find the optimal parameter settings under user's accuracy requirement, without or with considering user's budget requirement. We prove that the parameter setting generated by IGD/TIGD algorithm form a Nash Equilibrium, and also suggests an accuracy lower bound. Our experiments show that even in the most severe situation, where the malicious workers dominate the outsourced computing environment, our algorithm is able to find the parameter settings satisfying user's budget and accuracy requirement.

1. Introduction

Outsourced computing is gaining popularity in recent years. The essential reason behind such a booming technical trend is the challenge of ever increasing computation scale and difficulty.

The increase of computing scale is substantiated by the computations whose input data sizes are significantly large, known as "big data". The solution to such a challenge is to adopt parallel computation paradigm, such as MapReduce [1]. The parallel computation framework is usually running as a cluster, which consists of multiple worker nodes. During computation, the input is split into multiple chunks, each of which is assigned to a worker node and processed as a task. Such a framework could easily scale up by adding more computation nodes to the cluster if the input size increases. However, since the customer does not want or cannot afford of investing a computation cluster, she usually

* Corresponding author at: School of Computer Science and Technology, Xidian University, 2 South Taibai Road, Xi'an, Shaanxi, PR China.
E-mail addresses: yongzhiwang@icloud.com (Y. Wang), weijp@cs.fiu.edu (J. Wei), sren@ece.ucr.edu (S. Ren), ylshen@mail.xidian.edu.cn (Y. Shen).

http://dx.doi.org/10.1016/j.future.2015.08.010
0167-739X/

outsources her computation to a computation service provider (e.g., cloud vendor[1] or grid computing network [2,3]), on which she can create a cluster swiftly and on demand, while saving the costs of infrastructure set up and maintenance.

The challenge of computation difficulty is substantiated by the fact that computers cannot replace humans on tasks that require human intelligent, such as recognizing images, finding bushiness information on the Internet, or solving Captchas. In most cases, such a class of computation is not difficult for human being. Yet they are tedious and require a lot of human effort. Crowdsourcing such as Mechanical Turk[2] offers an option to address such a challenge. In this computing paradigm, the crowdsourcing company categorizes similar tasks into batches, and assigns tasks to humans on the internet to solve. The crowdsourcing company receives the task results from the workers, verifies the correctness of their results and pays workers based on the correctness of their task results.

Although outsourced computations can have a variable of different formats, they share the following similar characteristics.

(a) The computation job is usually significantly large, so that the job owner cannot process it locally. Therefore, such jobs usually are split into multiple tasks, and computed by computation service providers.
(b) The task number of a job is usually proportional to the input data size and is usually large. When the input data size increases, the job can scale up by increasing the task number.
(c) The outsourced computation is usually not free. The computation service provider usually charges the user based on the amount of computation performed (correctly).

However, most outsourced computing paradigms share the same vulnerability: *Since the tasks are outsourced to untrusted workers, if some workers are malicious, they could tamper the task result, and therefore affect the job result correctness.*

In this paper, we extend the existing solution in [4] and [5] to a generalized class of outsourced computing jobs, propose *Integrity Assurance Outsourced Computing* (*IAOC* for short) system. IAOC employs probabilistic task replication, probabilistic task verification and credit management techniques to offer high result accuracy assurance. We perform theoretical analysis on IAOC system and model the accuracy of outsourced computing based on the system parameters. We find that the outsourced computing accuracy is determined together by IAOC system and the participant malicious workers. Hence, we model the outsourced computing as a two-player zero sum game, where the IAOC system and the malicious workers can adjust their behaviors to maximize their benefits. We propose a novel algorithm called *Interactive Gradient Descent* algorithm (*IGD* for short) to search for the optimal behaviors (i.e., parameter setting) for IAOC system that satisfies user's accuracy requirement. We prove that the algorithm generated optimal behavior forms Nash Equilibrium. In other words, either the IAOC and the attacker do not have incentive to deviate from the algorithm suggested behavior. Therefore, the algorithm predicted accuracy is the highest lower bound.

Based on IGD algorithm, we propose a *Tiered Interactive Gradient Descent* (*TIGD*) algorithm, that also generates optimal behavior for IAOC system and the malicious worker, while considering user's outsource budget from multiple aspects (see Section 5.2). The generated optimal behavior satisfies user's budget requirement, meanwhile, guarantees the highest lower bound of the job accuracy.

We perform a set of experiments based on TIGD algorithm. The result shows that the algorithm is effective in finding optimal parameter setting even in the most severe situation where all the workers in the outsourced environment are malicious.

The rest of this paper is organized as follows. Section 2 defines the generalized outsourced computation system and the system assumptions. Section 3 presents the integrity assurance outsourced computing (IAOC) system on the generalized outsourced computing system. Section 4 builds a mathematical model of IAOC system to measure the accuracy of the job and performs a set of simulation based on this model. Section 5 presents the IGD and TIGD algorithm. Section 6 describes and analyzes the experiment result. Section 7 discusses related work, and Section 8 concludes the paper.

2. System definition and assumption

2.1. System definition

In this section, we define the generalized outsourced computing system. The entity who provides the outsourced computing service is called a *provider*. The entity who outsources its computation to the provider is called a *user*. A provider maintains a distributed system to perform the computation. The system consists of a *master* and many *workers*. The master controls the entire computation and responsible for assigning computation to workers. Each worker is the actual entity to perform the computation. The unit of computations outsourced by the user is a *job*. Since the outsourced job is usually significantly large, a job is usually split into multiple (maybe a significant number of) *tasks*, which are assigned to different workers to compute. As a reward, the provider charges the user according to the amount of computation she has performed for that user. The user can verify the task results returned by the provider. If the user finds that the task results are incorrect, she can reject the provider's results, but she has to show the proof to the provider that the results are incorrect. Therefore, the user only pays the provider the amount of computation that she accepts.

In some scenarios, the master accepts jobs from the user and hires workers to compute those jobs. For example, in crowdsourcing computing, the crowdsourcing company accepts jobs from the user, assigns tasks to workers on the Internet and pays workers on behalf of the user. The crowdsourcing company is trusted. Therefore, the crowdsourcing company can verify the task results for the user and determine whether to accept or reject the worker submitted results.

2.2. System assumption

In the outsourced computing system, we assume the master is trusted, but workers can be malicious due to the open environment. The malicious workers' goal is to insert incorrect results to the job without been detected. For example, in the crowdsourcing computing scenario, the worker can return trivial but incorrect answers to the user in order to earn more money. We assume the malicious workers are highly intelligent. For example, they can exchange the task information and coordinate with each other to cheat at the optimal time. For instance, if two tasks that compute the same input are assigned to two malicious workers simultaneously, they can return the same erroneous results (i.e., to collude) so that simply comparing the task result results cannot detect the error. We call such malicious workers *collusive* workers.

We assume that the task number in a job is big enough and all task results are equivalently important to the overall job result. In other words, if a small number of task results are incorrect and undetected, it will not undermine the overall job accuracy

[1] Amazon Web Services. http://aws.amazon.com.
[2] https://www.mturk.com.

significantly. We argue that this is a reasonable assumption. Firstly, one primary reason for the computation outsource is that the computation task number is too large. Otherwise, the user would perform the computation locally. Secondly, highly parallel is a necessary factor to address the challenge of scaling-up input data. In such a highly parallel computations, task results are usually equally important to the overall job result. However, we have to point out that for some jobs, the importance of task results are severely skewed. Our method is not suitable for such a class of jobs.

3. Integrity assurance system design

We expend the MapReduce integrity assurance framework proposed in [4] to the generalized outsourced computing system and propose the Integrity Assurance Outsourced Computing (IAOC) system. The system work flow is shown in Fig. 1.

In IAOC algorithm, each worker will return the task result (or the hash code of the result) to the master when she finishes a task. The master compares or verifies the returned results to detect malicious workers. During the job execution, IAOC performs a *two-layer result checking* technique on each task. In the first layer check, the master firstly assigns the task to a randomly chosen worker $W1$ (step 1) and asks the worker to return its task result (or hash code of the result) (step 2). Then, the master probabilistically replicates this task (step 3) and assigns the replicated task to another randomly chosen workers $W2$ (step 4). The probability that a task being replicated is called the *replication probability*, marked as r. Notice that the master replicates a task and assigns the replicated task *only* when its original task result is returned to the master. We call this technique as *hold-and-test*. Such a sequence is to reduce the malicious workers' collusion chance. When the original and replicated task results are returned, the master compares the two results to decide if a malicious worker exists (step 5). If the task results are different, the master will verify the task by re-executing the task itself and determine the malicious worker (step 6). If the task results are the same, the task has passed the first layer check and the master will perform the second layer check on it. The second layer check is used to detect the collusive worker. In the second layer check, the master verifies the consistent task result probabilistically (step 7). The probability of verifying a pair of consistent results is called *verification probability*, marked as v.

Since in the two-layer check, the task replication and the result verification are performed probabilistically, the un-replicated/un-verified task result can be incorrect and undetected. In order to guarantee a high probability of task result correctness, the master maintains a credit for each worker and only accepts a worker's task results in a batch when the worker's credit achieves certain threshold. Before that, the master stores the worker's task results in that worker's buffer temporarily. For example, when a task executed by worker $W1$ passes a two-layer check, the master increments $W1$'s credit (step 9) and stores the original task result in $W1$'s buffer. When the credit of a worker achieves certain threshold, the worker becomes trusted temporarily and the buffered results generated by this worker is accepted in a batch by the master (step 10). The buffer is therefore emptied. The threshold is called the *credit threshold*, marked as T. After the results are accepted, this worker becomes untrusted again, and its credit is reset to 0. If a worker fails at any two-layer check, the master shows that worker the proof of the check results (step 11) and rejects the task results stored in that worker's buffer. The master then moves it to a black list (step 12) and will never assign tasks to it. For the tasks that are rejected by the master, the master drops those results returned by that worker and reschedules those tasks (step 13).

In the IAOC system, the master is trusted. Therefore the master checks the correctness of the task results and pays the workers on behalf of the user. The master pays each worker based on the number of tasks whose results are accepted by the master. The master does not pay for a task if its result is rejected by the master. With such a policy, even if tasks can be rescheduled due to the detection of malicious workers, the master only pays each original task once when its result is accepted. (The master also needs to pay for the replicated task if the first-layer check is performed.) The master does not have to pay for the rejected tasks. Therefore task rescheduling will not increase the master's financial burden. For example, in the crowdsourcing computing case, the crowdsourcing company only pays the workers for the tasks whose results are accepted. The company does not pay for those rejected tasks.

Note that in a real business scenario, the master usually pays the workers on behalf of the user. Therefore, the master will charge the user the same amount of money it pays the workers. Since the master only has to pay the worker for the tasks that it accepted, the cost of this part is proportional to the number of original tasks and their replication tasks if available. In other words, the cost is proportional to the job size and the replication probability r. In addition, the master also has to charge the user for the cost of job management, e.g., the cost of the verification tasks performed in the second-layer check. Since each original task will be verified at most once, the cost for the verification is proportional to the verification probability v. Therefore, the cost for a user to outsource a job to IAOC is proportional to the number of original tasks, the replication probability r and the verification probability v.

We perform the theoretical analysis in Section 4 and propose algorithms to select the optimal system parameter values for T, r and v to achieve the highest job accuracy lower bound, without or with considering user's budget restriction.

4. System model and simulation

4.1. System model

In this section, we build a mathematical model to evaluate the job accuracy of the IAOC algorithm. Since we assume the outsourced computing is highly parallel and each task result is equally contributed to the overall job result, the job accuracy is proportional to the portion of correct task results. Therefore, we define the *job error rate* as the fraction of incorrect task results out of the total task results accepted in one job, marked as J.

Before creating a model to calculate J, we need to model malicious workers' behavior.

We define the fraction of malicious workers in the outsourced environment as the *malicious worker fraction*, marked as m. We assume m to be a constant value. That is, even though some malicious workers are detected and black-listed during job execution, m decreases very little either because the total worker number is large, or because the existence of sybil attack [6].

Since it is easier to detect the non-collusive worker than the collusive worker, we consider the worst situation, that is we assume all the malicious workers in the outsourced environment are collusive workers. Due to the existence of hold-and-test technique, our following analysis shows that the best strategy for the collusive worker is to cheat randomly in a hope of not being found.

Remember that the malicious worker's goal is to inject as many incorrect task results as possible without being detected. We are now performing adversary analysis to analyze malicious workers' strategy under the IAOC system.

The adversary analysis

Suppose a task is assigned to a malicious worker, let us say W, this task can be an original task or a replicated task. We analyze W's strategy on the two cases.

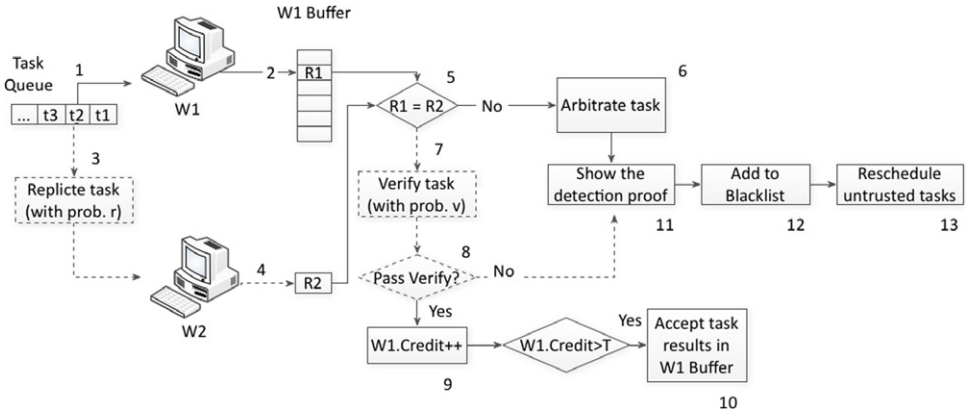

Fig. 1. Integrity assurance outsourced computing.

(1) If the task is an original task, W does not know whether the task will be replicated. Also, if the task will be replicated, W also does not know which worker will be assigned to execute the replicated task. W is therefore in a dilemma to decide whether to cheat. If she decides not to cheat, she loses a chance of inserting incorrect results to the job. If she decides to cheat, she can be detected if the task is later replicated and assigned to a benign worker. Even if the replicated task is assigned to a collusive worker later, W can still be detected if the task result is verified in the second-layer check. Therefore, in this case, in order to inject error results, W can only cheat randomly in a hope of not being detected.

(2) If the task is a replicated task, W can first query other malicious workers to see if the corresponding original task was executed by a malicious worker.
   (a) If a malicious worker, let us say $W\prime$, replies that she has executed the original task, W will return the master a result consistent to what $W\prime$ returned.
   (b) If no malicious worker responds the inquiry, W knows that the original task was executed by a benign worker. Thus W will honestly execute the task and return the correct result to the master.

In fact, W can only determine whether the current task is a replicated task in case 2(a) (by sending inquiry to other malicious workers). She cannot distinguish case 1 and case 2(b). Therefore, for W, it is only safe to cheat in case 2(a). In other cases, in order to inject error results, she can only cheat randomly in a hope of not being detected.

Based on the above analysis, the strategy for a malicious worker to perform is as follows.

The collusive worker's strategy

When a collusive worker, W, receives a task, she will first test whether it is a replicated task: she assumes this task to be a replicated task and query other malicious workers to see if anyone has executed its corresponding original task.

(1) If another malicious worker, $W\prime$, replies. W knows that the current task is a replicated task and the original task was executed by $W\prime$. W then returns the master a task result consistent to what $W\prime$ returns.

(2) If no malicious worker responds the inquiry, W knows that either the current task is an original task, or it is a replicated task whose corresponding original task was executed by a benign worker. But she cannot distinguish the two situations. Therefore, W randomly determines whether to cheat and returns a correct or incorrect result to the master according to the decision.

Since W's cheat behavior in case 2 is randomized, we model this behavior as a probabilistic event. We define the probability that a collusive worker cheats as the *cheat probability*, marked as c.

We model the IAOC system with the replication probability r, verification probability v, and credit threshold T, as presented in Section 3. We summarize the system parameters and evaluation metrics in Table 1. We perform probability analysis and model the job error rate, shown in Theorem 4.1.

Theorem 4.1. *Assuming that the task assignment is performed independently by the master and uniformly distributed across all workers on the IAOC environment, the probability for a malicious worker to survive after executing n original tasks is*

$$S_n = (1 - cr + crm(1 - v))^n. \tag{1}$$

The job error rate of IAOC is

$$J = m(c(1 - r) + crm(1 - v))(1 - cr + crm(1 - v))^{T-1}. \tag{2}$$

The proof of Theorem 4.1 is shown in the Appendix

4.2. System simulation

In order to obtain an intuition on how the system parameters affect the job error rate J, we perform a set of simulations based on (2). The simulation results are shown in Fig. 2.

Fig. 2(a) shows that when other parameters (i.e., c, r, v and m) are fixed, increasing credit threshold T will reduce job error rate J. When T is small, increasing T can reduce J quickly. Yet as T grows, the effect of reducing the job error rate is weakened. For example, when m is 1.0, increasing T from 0 to 200 can reduce J from 0.1 to 0.04. However, when T is further increased from 200 to 400, J is only reduced from 0.04 to 0.015. The figure also shows that when T is fixed, a higher value of m can incur a higher value of job error rate.

Fig. 2(b) and (c) shows that when other parameters (i.e., T, c, m and v (or r)) are fixed, increasing r (or v) can also help reducing job error rate. Similarly, J reduces faster when r (or v) is small than when r (or v) is large. The two figures also show that a higher value of m incurs a higher job error rate when other parameter values are fixed.

Fig. 2(d) shows how the job error rate changes with different cheat probability c. From the malicious worker's perspective, in order to achieve the highest job error rate, the malicious worker has to make a trade-off and choose an optimal value for c. If c is too high, the malicious worker can be easily detected. If the c is too low, very few incorrect task results will be inserted. As Fig. 2(d) shows, when T is 100, r is 0.3, v is 0.15 and m is 1.0, setting c close to 0.2

Table 1
Integrity assurance outsourced computing modeling parameters.

Note	Item	Definition
m	malicious worker fraction	The fraction of malicious workers in the outsourced environment.
c	cheat probability	The probability that a malicious worker decides to cheat if she finds she is not in case 2 in the Collusive Worker's Strategy.
r	replication probability	The probability that IAOC system replicates a task in the first-layer check.
v	verification probability	The probability that IAOC system verifies a task in the second-layer check.
T	credit threshold	The credit a worker has to achieve to make his task results accepted by the master.
J	job error rate	The fraction of incorrect task results out of the total task results accepted in one job.

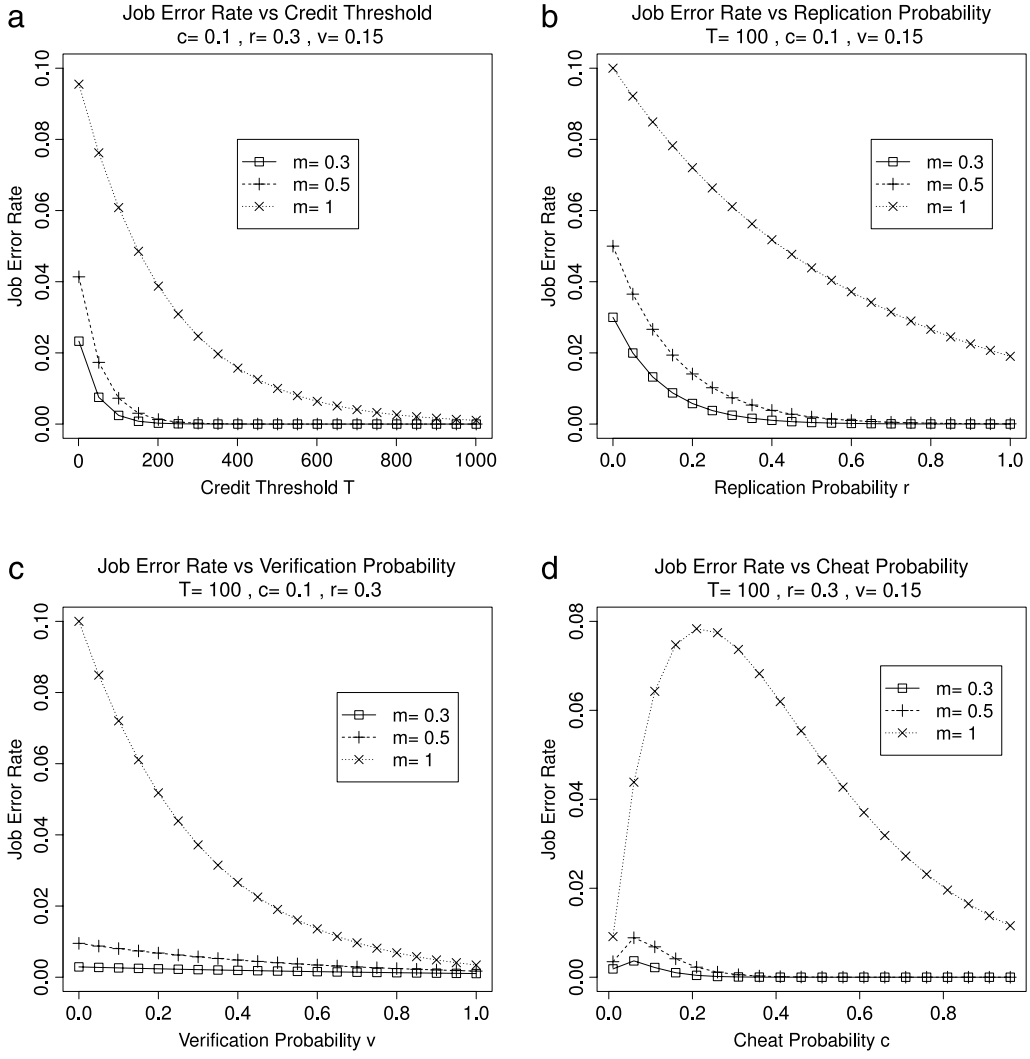

Fig. 2. Simulation results of IAOC algorithm under different parameters.

will achieve the highest job error rate (less than 0.08). Similarly, when other parameters are fixed, increasing m will incur a higher job error rate.

In fact, it is easy to prove the following monotonic property for job error rate (2) by analyzing the signs of the partial derivatives of (2) on T, r, v, m and c, respectively:

Theorem 4.2. *When other parameters are fixed, the job error rate J will*

(a) *monotonically decrease if the credit threshold T increases.*
(b) *monotonically decrease if the replication probability r increases.*
(c) *monotonically decrease if the verification probability v increases.*
(d) *monotonically increase if the malicious worker fraction m increases.*
(e) *first increase and then decrease when the cheat probability c increases.*

We also study the monotonicity of the job error rate (2) when the attacker always chooses the best cheat probability.

Theorem 4.3. *When other parameters are fixed and the attacker always chooses the optimal cheat probability to maximize the job error rate, the job error rate J will*

(a) *monotonically decrease if the credit threshold T increases.*
(b) *monotonically decrease if the replication probability r increases.*
(c) *monotonically decrease if the verification probability v increases.*

Proof. Since J is a convex function on variable c, the attacker can search for the optimal value of c to maximize the job error rate by solving the equation $\frac{\partial J}{\partial c} = 0$. By solving the equation, the attacker has

$$c = \frac{1}{rT(1 - m(1 - v))}. \tag{3}$$

When the attacker sets c as $\frac{1}{rT(1-m(1-v))}$, it will make the IAOC system achieve the highest job error rate. By substituting c with (3) on (2), we have

$$J = m\frac{(T-1)^{T-1}}{T^T}\left(\frac{1}{r(1-m+mv)} - 1\right). \tag{4}$$

According to (4), J is monotonically decreasing, when either $T > 1$ is increasing, or r is increasing, or v is increasing.

Theorem 4.2 suggests that the IAOC system and the attacker together determine the job error rate.

From the IAOC system's perspective, the job error rate will be decreased if she

(a) Increases the credit threshold T, or
(b) Increases the replication probability r, or
(c) Increases the verification probability v.

From the Attacker's perspective, the job error rate will be increased if she

(a) Increases the malicious worker fraction m, or
(b) Sets the cheat probability c to the optimal value to achieve a maximum job error rate.

The system forms a two-player zero-sum game. In the next section, we propose two algorithms to find system parameter settings that suggest optimal parameter setting that satisfies the user's accuracy requirement, without or with considering user's budget requirement.

5. Finding optimal parameter setting

In this section, we introduce two algorithms to find the optimal system parameter setting. We first introduces the *Interactive Gradient Descent Algorithm* (*IGD* algorithm for short), which automatically searches the parameter setting of T that can guarantee user's accuracy requirement no matter how does the attacker sets her cheat probability c.

Based on IGD algorithm, we describe *Tiered Interactive Gradient Descent* (*TIGD*) Algorithm, which automatically searches the parameter settings that make the job error rate J as close to the user's accuracy requirement as possible, meanwhile satisfying user's budge requirements.

5.1. Interactive Gradient Descent Algorithm

In this section, we propose *Interactive Gradient Descent Algorithm*, an algorithm that can search for the optimal value of T to satisfy user's accuracy requirement. In this algorithm, we assume the values of r and v are preset by the user and are fixed. We present the Tiered Interactive Gradient Descent algorithm in Section 5.2 to remove this assumption.

According to Theorem 4.2, from the IAOC system's perspective, setting T as large as possible surely can minimize the job error rate. However, setting T as a significantly large number of tasks requires the accumulation of a significantly large credit, which is not practical in the real situation. On the other hand, the effect of decreasing the job error rate is weakening when T grows large. Therefore, we assume the accuracy is acceptable if IAOC system can achieve a reasonably small job error rate. We define such a job error rate as the *acceptable job error rate*, marked as J_0. Usually, J_0 is a very small number that is greater than 0 but very close to 0. Our goal is to find a value T that makes the job error rate close enough to J_0 when malicious workers choose the optimal cheat probability to maximize the job error rate.

In order to achieve our goal, we analyze the behavior of both IAOC and the attacker. Since the IAOC and the attacker form a two-player zero-sum game, the two players have conflicting interests. For the attacker, it will maximize the job error rate J (i.e., to make J as close to 1.0 as possible). For IAOC, it will make the job error rate close to the acceptable job error rate J_0 as much as possible. We formalize the two players' interests by defining their loss functions. We show that finding a parameter setting that minimizes a player's loss function is equivalent to achieving that player's goal. In other words, for each player, in order to achieve its goal, it will search for a parameter setting that minimizes its loss function.

For the IAOC system, we define its loss function as follows.

$$L_{IAOC} = (J - J_0)^2. \tag{5}$$

By substituting J with (2), we have

$$L_{IAOC}(T, r, v, m, c, J_0) = \Big(m\Big(c(1-r) + crm(1-v)\Big) \cdot \Big(1 - cr + crm(1-v)\Big)^{T-1} - J_0\Big)^2.$$

The squire operation in (5) ensures that as J approaches J_0, the loss function becomes smaller. In other words, if IAOC can minimize its loss function, she can make J as close to J_0 as possible, which achieves its goal. Since we assume the value of r and v are fixed, to minimize the loss, IAOC needs to search for an appropriate value for T.

On the other hand, when IAOC chooses a value for T, r and v, the attacker will need to search for appropriate values for m and c to maximize the job error rate, and thereby to achieve its goal. For the value of m, the attacker only needs to set it as large as possible, since a larger value of m will incur a higher job error rate. Therefore, we assume m to be the highest fraction of malicious workers the attacker can inject to the public cloud. In the worst case that the malicious worker dominates the public cloud, we assume m as 1.0. For the value of c, the attacker needs to find a proper value between 0 and 1 to maximize the job error rate (according to property (e) in Theorem 4.2). We define the attacker's loss function as follows:

$$L_{ATT} = (1 - J)^2. \tag{6}$$

By substituting J with (2), we have

$$L_{ATT}(T, r, v, m, c) = \Big(1 - m\Big(c(1-r) + crm(1-v)\Big) \cdot \Big(1 - cr + crm(1-v)\Big)^{T-1}\Big)^2.$$

The square operation in (6) ensures that as J approaches 1.0, the loss function becomes smaller. In other words, if the attacker can minimize its loss function, she can make J as large as possible, which achieves her goal. In order to minimize the loss function, the attacker has to find a suitable value for c.

Both IAOC and the attacker need to minimize their own utilities. However, the value of the loss function for each player is determined by the parameters controlled not only by itself but also by its opponent. We therefore propose the Interactive Gradient Descent algorithm that can help IAOC search for a parameter setting, which makes both parties have no incentive to deviate. In other words, the output of the algorithm achieves Nash Equilibrium in the two-player game. The algorithm is shown in Algorithm 1. The intuition of the algorithm is to simulate the two-player game, in which two players take turns to adjust the parameter it controls to minimize its loss function value. Theorem 5.1 shows that after limited iterations, the algorithm will terminate. Theorem 5.2 shows that the algorithm result ensures the game achieves Nash Equilibrium. Corollary 5.2.1 shows that the algorithm returned job error rate will be close enough to the acceptable job error rate J_0.

Algorithm 1 Interactive_Gradient_Descent (m, r, v, J_0, T_{init}, c_{init})

1: $T \leftarrow T_{init}$
2: $c \leftarrow c_{init}$
3: $J_{IAOC} \leftarrow 0$
4: $J_{ATT} \leftarrow 1$
5: **while** $|J_{IAOC} - J_{ATT}| < \delta$ **do**
6: $result \leftarrow minimizeLiaoc(L_{IAOC}, T, r, v, m, c, J_0)$
7: $J_{IAOC} \leftarrow result.getNewJ()$
8: $T \leftarrow result.getNewT()$
9: $result \leftarrow minimizeLatt(L_{ATT}, T, r, v, m, c)$
10: $J_{ATT} \leftarrow result.getNewJ()$
11: $c \leftarrow result.getNewC()$
12: **end while**
13: $T_{opt} \leftarrow T$
14: $c_{opt} \leftarrow c$
15: $J_{opt} \leftarrow J_{ATT}$
16: **return** $T_{opt}, c_{opt}, J_{opt}$

As shown in Algorithm 1, the IGD algorithm takes accepted job error rate J_0, m, r, v, the initial value of T (T_{init}) and the initial value of c (c_{init}) as arguments and returns a suggested value of T (T_{opt}), which guarantees that the job error rate does not exceed J_{opt}, no matter what cheat probability the attacker chooses (in fact, the optimal value of c the attacker can choose to maximum the job error rate is the suggested c_{opt}). Note that in this algorithm, m, r and v are fixed values, which do not change during the algorithm execution.

The algorithm performs multiple iterations until converge. In each iteration, it first searches for the value T that minimizes the loss function L_{IAOC} on variable T, with the current value c fixed (in the function *minimizeLiaoc* in line 6 of Algorithm 1). This step simulates the situation that when the attacker sets a value for c, IAOC searches for the optimal value of T that makes J close to J_0 as much as possible. After this step, the corresponding value of J is recorded in J_{IAOC}, and the value of T is updated with the new value (lines 7 and 8). After that, it invokes the function *minimizeLatt* (line 9) to search for a value of c that minimizes the attacker loss function L_{ATT}. This step simulates the situation that when IAOC sets a value for T, the attacker searches for the optimal value of c that makes J as high as possible (i.e., as close to 1 as possible). After this step, the corresponding value of J is recorded in J_{ATT}, and the value of c is updated with the new value (lines 10 and 11). After each iteration, the algorithm checks if the difference between J_{IAOC} and J_{ATT} are reduced to a small value (i.e., the variable δ). If true, the iteration terminates and the latest value of T, c and J_{ATT} are returned; otherwise, the algorithm proceeds to the next iteration.

Since L_{IAOC} is differentiable on variable T, we can perform gradient descent on variable T in function *minimizeLiaoc* to minimize L_{IAOC}. Since L_{ATT} is differentiable on variable c, we can perform gradient descent on variable c in function *minimizeLatt* to minimize L_{ATT}. Gradient descent algorithm can search for the local minimum value for a given function. Since the two loss functions are both convex functions, whose local minimum value is also the global minimum value, the return of the gradient descent algorithm will always be the optimal parameters that minimize the loss. The two function implementations are standard gradient descent procedure. The pseudo code of *minimizeLiaoc* is shown in Algorithm 2. The algorithm adjusts the value of T in the negative direction of the loss function gradient in multiple iterations until the loss function value converges. The same operation is performed on the variable c for the loss function L_{ATT} in the function *minimizeLatt*. Due to the space limitation, we skip the pseudo code here. The accuracy and the efficiency of the IGD algorithm depends on three parameters: the step size γ and the convergence threshold ϵ in Algorithm 2 and the δ in Algorithm 1. If the step size γ is too small, the convergence of gradient descent algorithm will be very slow. If the step size γ is too large, the convergence will be initially very fast, but the algorithm will oscillate about the optimum later. The convergence threshold ϵ in Algorithm 2 and δ in Algorithm 1 should be set as small as possible. However, setting the values too small will make the convergence very slow. We chose reasonable values for the above parameters based on experimental results. We will discuss the choice of all the parameters in Section 6.

Algorithm 2 minimizeLiaoc($L_{IAOC}, T_{init}, r_{init}, v_{init}, m_{init}, c_{init}, J_0$)

1: $loss_{old} \leftarrow +\infty$
2: $loss_{new} \leftarrow L_{IAOC}(T_{init}, r_{init}, v_{init}, m_{init}, c_{init}, J_0)$
3: $T \leftarrow T_{init}$
4: **while** $|loss_{new} - loss_{old}| > \epsilon$ **do**
5: $T \leftarrow T - \gamma * \frac{\partial L_{IAOC}}{\partial T}$
6: $loss_{old} \leftarrow loss_{new}$
7: $loss_{new} \leftarrow L_{IAOC}(T, r_{init}, v_{init}, m_{init}, c_{init}, J_0)$
8: **end while**
9: **return** T

The following theorems show that, Algorithm 1 will converge after a limited number of iterations; the return parameter values guarantee Nash Equilibrium and the J_{opt} returned from the algorithm is close enough to J_0.

Lemma 5.0.1. *For any two consecutive iterations in Algorithm* 1*, suppose the algorithm generates J_{IAOC_i} and J_{ATT_i} in an iteration, and generates $J_{IAOC_{i+1}}$ and $J_{ATT_{i+1}}$ in the next iteration. We have*

$$0 < J_{IAOC_{i+1}} < J_{ATT_{i+1}} < J_{ATT_i}. \tag{7}$$

Proof. It is trivially to see that $0 < J_{IAOC_{i+1}} < J_{ATT_{i+1}}$. In (5), when J_0 is small enough, $L_{IAOC}(T)$ is monotonically decreasing with the increase of T. Therefore, in any two consecutive iterations in Algorithm 1, namely iteration i and iteration $i+1$. We mark the resulting J_{IAOC}, J_{ATT}, T and c in iteration i as J_{IAOC_i}, J_{ATT_i}, T_i and c_i. In order for the iteration $i+1$ to receive a minimum job error rate after calling *minimizeLiaoc* at line 6 of Algorithm 1, T will be increased. In other words, $T_{i+1} > T_i$. Since J is monotonically decreasing with the increase of T (according to Theorem 4.2), which makes the minimum of L_{ATT} monotonically increase with the increase of T, we have $Min(L_{ATT_i}) < Min(L_{ATT_{i+1}})$. Thus $J_{ATT_{i+1}} < J_{ATT_i}$.

Theorem 5.1. *Algorithm* 1 *will terminate after a limited number of iterations.*

Proof. According to Lemma 5.0.1, J_{ATT} is monotonically decreasing in each iteration. We are proving that after limited number of iterations, $|J_{IAOC} - J_{ATT}|$ will be no larger than an arbitrarily small positive number δ. By contradiction, we assume $|J_{IAOC} - J_{ATT}|$ is always larger than a small value δ, since J_{ATT} is monotonically decreasing, suppose after k iterations, J_{ATT} decreases to $J_{ATT_k} < \delta$, since $J_{IAOC_k} < J_{ATT_k}$ and $|J_{IAOC_k} - J_{ATT_k}| < \delta$, we have $J_{IAOC_k} < 0$, which contradicts with Lemma 5.0.1. Therefore, setting the threshold in Algorithm 1 as δ will make the algorithm terminate. Since δ is an arbitrarily small positive number, setting threshold as any positive value will guarantee the termination of Algorithm 1.

We show that the resulting algorithm is a Nash equilibrium.

Theorem 5.2. *The Algorithm* 1 *returns a set of parameter setting that form Nash Equilibrium.*

Proof. Firstly, since the loss functions L_{IAOC} and L_{ATT} are convex functions. They both have no more than one minimum value. Therefore, the local minimum values generated by the gradient descent algorithm are global minimum values.

Suppose the IGD algorithm returns J_{opt}, c_{opt} and T_{opt}. According to the definition of the algorithm, T_{opt} ensures that the IAOC system's loss function L_{IAOC} achieves the global minimum value, when the attacker sets the cheat probability as c_{opt}. Also, c_{opt} ensures that the attacker's loss function L_{ATT} achieves the global minimum value, when IAOC system sets the credit threshold as T_{opt}. Therefore, for IAOC system, changing T from T_{opt} to any other value T' will increase L_{IAOC}, if the attacker keeps the setting of c as c_{opt}. The IAOC system therefore has no incentive to deviate from T_{opt}. Similarly, for the attacker, changing c from c_{opt} to any other value c' will increase L_{ATT}, if IAOC system keeps the setting of T as T_{opt}. The attacker therefore has no incentive to deviate from c_{opt}. Since both players have no incentive to deviate from IGD returned parameter values, Nash Equilibrium is achieved.

Corollary 5.2.1. *The IGD algorithm (Algorithm* 1*) returned job error rate J_{opt} satisfies user's accuracy requirement. In other words, J_{opt} is close enough to J_0.*

Proof. Suppose Algorithm 1 terminates when the value of J_{IAOC} is close enough to J_{ATT} at a certain iteration, i, we have $|J_{IAOC_i} - J_{ATT_i}| < \delta$. According to Theorem 4.2, J is monotonically decreasing when T increases. It is also easy to prove that $\lim_{T\to\infty} J = 0$. Therefore, for any J_0 that is close to 0, there exists a T that makes $J = J_0$. In other words, the minimum of L_{IAOC} is 0. A gradient descent algorithm performed in Algorithm 2 can find a local minimum of L_{IAOC}, which is a global minimum due to the fact that L_{IAOC} is a convex function. Therefore the invocation of *minimizeLiaoc* at line 6 of Algorithm 1 returns an J_{IAOC_i}, which makes $|J_{IAOC_i} - J_0| < \xi$, where ξ is an arbitrary small positive number. Since $J_{opt} = J_{ATT_i}$, we have $|J_{opt} - J_0| < \xi + \delta$. Since ξ and δ are both arbitrary small values, J_{opt} is close enough to J_0.

Since the IGD algorithm returned job error rate is close enough to the accepted job error rate and the attacker cannot further increase the job error rate, the parameter setting returned from the IGD algorithm is the optimal setting. By setting the credit threshold T with the IGD returned value T_{opt}, IAOC can achieve the lowest job error rate upper bound J_{opt}.

5.2. Tiered interactive gradient descent algorithm

Normally, the user wishes to set J_0 as a very small value (e.g., 0.001), and assumes m is large (e.g., 1.0). Our experiment in Section 6 shows that under such a setting, the IGD algorithm will return a large value of T. For example, according to the last row of Table 3, when m is 1.0, J_0 is 0.001, the IGD algorithm returns a T value as large as 27,796. It is because the effect of decreasing job error rate is weakening when T increases. (see Fig. 2(a)). However, we observe that increasing the value of v or r can effectively decreasing job error rate, especially when the value of v or r is small (see Fig. 2(b) and (c)). Therefore, in order to achieve a small job error rate, instead of unlimitedly increasing T, increasing v or r would be a good option.

On the other hand, outsourced computing is not free. Users usually have to consider budget restrictions when outsourcing jobs. Budget restrictions include the following aspects:

(a) The total number of tasks in a job. Even if increasing T can help decreasing job error rate, the IAOC system requires the total number of tasks no less than the value of T. Hence the credit threshold should be no greater than the total task number. It should have a limit. We call the maximum value of T that IAOC can set as the *maximum credit threshold*, marked as T_{max}.
(b) The number of replicated tasks. Since the computing service provider usually charges users according to the used resource, assuming each task takes the similar amount of resource, the extra financial cost is therefore proportional to the number of replicated tasks. Hence the replication probability r should have an upper limit. We call the maximum value of r that IAOC can set as the *maximum replication probability*, marked as r_{max}.
(c) The number of verified tasks. Verifying task requires users to consume their own computing resources, which may not be free (e.g., it requires users to invest on their own IT infrastructure) or time consuming (e.g., it needs to download data from computing service providers). Hence the verification probability v should also be bound with a limitation. We mark the maximum value of v that IAOC can set as the *maximum verification probability*, marked as v_{max}.

Under this situation, we propose the Tiered Interactive Gradient Descent (TIGD) algorithm, which ensures that the IAOC's budget is within the user's setting (T_{max}, r_{max} and v_{max}), meanwhile returning a job error rate as close to J_0 as possible.

The TIGD algorithm is based on the Interactive Gradient Descent algorithm. It performs three sets of IGD algorithms to adjust the value of T, r, and v, respectively. The sequence of the three interactive gradient descents is determined by the importance of the three budget aspects. The IGD on T and c is firstly performed because T can be increased "for free". As long as T does not exceed T_{max}, the user do not have to pay extra to the computing service provider. The IGD on r and c is performed before the IGD on v and c. It is because the task replication is performed on the public cloud, and the task verification is performed on the private cloud. Increasing task verification probability means putting more computing workload on the private cloud, which can increase IT infrastructure investment and involve more cross-cloud communication (due to the fact that the DFS is deployed on the public cloud). However, increasing replication probability only increases the workload on the public cloud, which is economically practical.

The TIGD algorithm is shown in Algorithm 3. The algorithm accepts the following parameters: the user evaluation on m (m_0), accepted job error rate (J_0); the initial replication probability and the maximum replication probability (r_{init} and r_{max}); the initial verification probability and the maximum verification probability (v_{init} and v_{max}); the initial credit threshold and the maximum credit threshold (T_{init} and T_{max}); and the initial value of the cheat probability (c_{init}). The algorithm returns a suggested parameter setting, T_{opt}, r_{opt}, v_{opt}, which guarantees that the job error rate does not exceed J_{opt}, no matter what cheat probability the attacker chooses (in fact, the optimal value of c the attacker can choose to maximize the job error rate is c_{opt}).

The algorithm performs three sets of iterations. Each set of iteration performs a round of IGD algorithm. However, in the IGD algorithm in Algorithm 1, the interactive gradient descents are performed on variables T and c. The three IGDs in Algorithm 3 are performed on variables T and c, r and c, and v and c, sequentially. Another difference is that in the TIGD algorithm, *restricted gradient descent algorithms* are performed instead of standard gradient descent algorithms on variables T, r, and v (in lines 8, 21, 35 in Algorithm 3). However, the standard gradient descents are still performed on variable c because we assume the attacker does not have any budget or restriction concerns. The restricted gradient descent on T is shown in Algorithm 4. In this algorithm, if the variable value achieves its upper bound before the loss function converges, the algorithm will terminate. For instance, when the restricted gradient descent is performed on variable T in Algorithm 4, the gradient descent algorithm terminates either when the loss function converges (in line 4), or when T achieves T_{max} (in line 6), whichever is earlier. The restricted gradient descent on r and v are similar to Algorithm 4, except that the variable to perform the gradient descent on r or v. We skip the pseudo code due to the space limit.

Algorithm 3 Tiered_Interactive_Gradient_Descent(m_0, J_0, r_{init}, r_{max}, v_{init}, v_{max}, T_{init}, T_{max}, c_{init})

```
1: T ← T_init
2: c ← c_init
3: r ← r_init
4: v ← v_init
5: J_IAOC ← 0
6: J_ATT ← 1
7: while |J_IAOC − J_ATT| < δ do
8:    gdResult ← restrictedMinimizeLiaocOnT(
9:                    L_IAOC, T, r, v, m, c, J_0, T_max)
10:   J_IAOC ← gdResult.getNewJ()
11:   T ← gdResult.getNewT()
12:   gdResult ← minimizeLatt(
13:                    L_ATT, T, r, v, m, c)
14:   J_ATT ← gdResult.getNewJ()
15:   c ← gdResult.getNewC()
16: end while
17: if |J_ATT − J_0| < δ then
18:   J_IAOC ← 0
19:   J_ATT ← 1
20:   while |J_IAOC − J_ATT| < δ do
21:      gdResult ← restrictedMinimizeLiaocOnR(
22:                    L_IAOC, T, r, v, m, c, J_0, r_max)
23:      J_IAOC ← gdResult.getNewJ()
24:      r ← gdResult.getNewR()
25:      gdResult ← minimizeLatt(
26:                    L_ATT, c, T, r, v, m)
27:      J_ATT ← gdResult.getNewJ()
28:      c ← gdResult.getNewC()
29:   end while
30: end if
31: if |J_ATT − J_0| < δ then
32:   J_IAOC ← 0
33:   J_ATT ← 1
34:   while |J_IAOC − J_ATT| < δ do
35:      gdResult ← restrictedMinimizeLiaocOnV(
36:                    L_IAOC, T, r, v, m, c, J_0, v_max)
37:      J_IAOC ← gdResult.getNewJ()
38:      v ← gdResult.getNewV()
39:      gdResult ← minimizeLatt(
40:                    L_ATT, c, T, r, v, m)
41:      J_ATT ← gdResult.getNewJ()
42:      c ← gdResult.getNewC()
43:   end while
44: end if
45: T_opt ← T
46: r_opt ← r
47: v_opt ← v
48: c_opt ← c
49: J_opt ← J_ATT
50: return T_opt, r_opt, v_opt, c_opt, J_opt
```

The Algorithm 3 first performs an interactive gradient descent on T and c (lines 7–16). This iteration will try to make J as close to J_0 as possible, meanwhile guaranteeing T not exceeding T_{max}. If the generated job error rate is not close enough to J_0, the algorithm will perform another interactive gradient descent on r and c (lines 20–29). In this iteration, T is set as a fixed value generated from the last IGD. This iteration will make J further closer to J_0, meanwhile guaranteeing r not exceeding r_{max}. If the resulting J is still not close enough to J_0, a final round of interactive gradient descent will be performed on v and c (lines 34–43). The resulting J_{opt} in line 49 is the final job error rate that is closest to J_0. Meanwhile, the resulting

Algorithm 4 restrictedMinimizeLiaocOnT(L_{IAOC}, T_{init}, r_{init}, v_{init}, m_{init}, c_{init}, J_0, T_{max})

```
1: loss_old ← +∞
2: loss_new ← L_IAOC(T_init, r_init, v_init, m_init, c_init, J_0)
3: T ← T_init
4: while |loss_new − loss_old| > ϵ do
5:    T_new ← T − γ * ∂L_IAOC/∂T
6:    if T_new > T_max then
7:       break
8:    end if
9:    T ← T_new
10:   loss_old ← loss_new
11:   loss_new ← L_IAOC(T, r_init, v_init, m_init, c_init, J_0)
12: end while
13: return T
```

T_{opt}, r_{opt} and v_{opt} will not exceed the user budget setting T_{max}, r_{max} and v_{max}, respectively.

Note that the TIGD algorithm does not guarantee that the resulting job error rate J_{opt} is arbitrarily close to J_0. If the values of T_{max}, r_{max} and v_{max} are too low, the resulting J_{opt} may not be close enough to J_0. In this situation, increasing T_{max}, r_{max} or v_{max} will be helpful in reducing the difference between J_{opt} and J_0. However, we can prove that J_{opt} is the lowest job error rate that IAOC can achieve under the current budget setting.

The following theorem proves that, the TIGD algorithm in Algorithm 3 will terminate after a limited number of iterations; the parameters suggested from the algorithm form Nash Equilibrium under the budget restriction; the job error rate J_{opt} returned from the algorithm is the lowest upper bound under the budget restriction.

Theorem 5.3. *Algorithm* 3 *will terminate after a limited number of iterations.*

Proof. In Theorem 5.1, we have proved that the interactive gradient descent in Algorithm 1 will converge. With the similar technique, we can prove that the IGD on r and c and IGD on v and c will converge.

The difference between the TIGD algorithm and the IGD algorithm is that the former uses restricted gradient descents in each set of iterations. Since the restricted gradient descent only adds one more termination condition for the current set of iteration, it only makes the current iteration terminate before the loss function value converges. Hence, it will not hinder the convergence at any round of IGD in the TIGD algorithm. Therefore, Algorithm 3 will terminate.

Theorem 5.4. *The TIGD algorithm (Algorithm* 3*) returned parameters, T_{opt}, r_{opt}, v_{opt} and c_{opt} form Nash Equilibrium under the budget restrictions.*

Proof. The TIGD algorithm (Algorithm 3) can terminate under three conditions:

(a) After the first IGD, the resulting job error rate is close enough to J_0 (i.e., the predicate $|J_{ATT} - J_0| < \delta$ in line 17 returns false);
(b) After the first IGD, the predicate in line 15 returns true. Also, after the second IGD, the resulting job error rate is close enough to J_0 (i.e., the predicate $|J_{ATT} - J_0| < \delta$ in line 31 returns false);
(c) The predicates in lines 17 and 31 both return true and the third IGD is performed.

From the attacker perspective, in each case, the resulting value of c in the last IGD will be the optimal value, which makes the attacker's loss function achieve minimum. Therefore, the attacker has no incentive to deviate from this value. From IAOC's perspective, it will end up with two possibilities:

Table 2
Parameter settings in TIGD experiments.

Item	Explanation	Value
J_0	The acceptable job error rate.	1E−3
γ_T	The step size in gradient descent on variable T. (The γ value in the implementation of *restrictedMinimizeLiaocOnT* in Algorithm 3.)	100
γ	The step size in gradient descent/restricted gradient descent on variable r, v and c. (The γ value in the implementation of *restrictedMinimizeLiaocOnR*, *restrictedMinimizeLiaocOnV* and *minimizeLatt* in Algorithm 3.)	5E−5
ϵ	The converge threshold in each gradient descent/restricted gradient descent (the ϵ value in the implementation of each minimization function in Algorithm 3).	1E−15
δ	The converge threshold in the TIGD algorithm. (The variable δ in Algorithm 3.)	1E−5
T_{init}	The initial value of T	10
r_{init}	The initial value of r	0.1
v_{init}	The initial value of v	0.1
c_{init}	The initial value of c	0.1

(1) The resulting J_{opt} is close enough to J_0. The possible terminate condition can be either a, b, or c.
(2) The resulting J_{opt} is not close enough to J_0. The possible terminate condition can only be c.

For possibility (1), the resulting loss function L_{IAOC} already achieves its minimum value. IAOC has no incentive to deviate from the parameters suggested from the TIGD algorithms. For possibility (2), the loss function L_{IAOC} does not achieve its minimum value. Increasing the value of T, r or v can further reduce the job error rate and thereby reducing L_{IAOC}. However, in this case, the algorithm returned T_{opt}, r_{opt}, and v_{opt} already achieve their maximum bounds, T_{max}, r_{max} and v_{max}, respectively. Further increasing those values will violate the budget requirement. Therefore, IAOC still has no incentive to deviate from T_{opt}, r_{opt} or v_{opt}.

Corollary 5.4.1. *The resulting value J_{opt} returned from Algorithm* 3 *either is close enough to the accepted job error rate J_0 or is the closest value to J_0 under budget restrictions.*

Proof. The algorithm will terminate with two possibilities:

(1) The returned J_{opt} is close enough to J_0. That is, $|J_{opt} - J_0| < \delta$.
(2) The returned J_{opt} is not close enough to J_0. That is, $|J_{opt} - J_0| >= \delta$.

For case (1), the returned J_{opt} is close enough to J_0. For case (2), the algorithm returned T_{opt}, r_{opt} and v_{opt} will achieve the maximum upper bounds T_{max}, r_{max} and v_{max}. By contradiction, if there exists another optimal job error rate $J'_{opt} < J_{opt}$, according to Theorem 4.3, J'_{opt}'s corresponding parameters T', r' and v' will satisfy at least one of the following conditions: $T'_{opt} > T_{opt}$, $r'_{opt} > r_{opt}$, $v'_{opt} > v_{opt}$, which will violate the budget restriction. Therefore, J_{opt} is the value closest to the acceptable job error rate J_0 under the budget restriction in case (2).

The TIGD algorithm can find parameter setting that either guarantees the resulting job error rate close enough to J_0, or guarantees the resulting job error rate to be the lowest upper bound within the budget restriction. Therefore, the TIGD algorithm returned parameter setting is an optimal setting. By setting system parameters as T_{opt}, r_{opt} and v_{opt}, IAOC can achieve the lowest job error rate upper bound.

6. Experiment

The execution of IAOC system consists of two phases: the parameter searching phase using the TIGD/IGD algorithm described in Section 3, followed by the actual job computation phase performing the two-layer checking technique described in Section 5. Since the two-layer checking technique is generalized based on the MapReduce integrity assurance framework proposed in [4], we refer readers to [4] for the experimental results of the actual job computation phase. In this section, we mainly study the effect of parameter searching phase. Specifically, we tested the effect of TIGD/IGD algorithm with different accuracy and budget requirements. Our experiments mainly focus on the TIGD algorithm. We treat the IGD algorithm as a special case of TIGD, where T_{max} is unbounded, $v_{max} = v_{init}$ and $r_{max} = r_{init}$. Since IAOC system is a generalized framework that is suitable for most outsourced computing scenario, our experiments are not targeting to a specific application. It is applicable to most outsourced computing scenario.

Table 3
Experiment result.

m	T_{max}	r_{max}	v_{max}	T_{opt}	r_{opt}	v_{opt}	J_{opt} (%)
1.0	10000	0.3	0.3	10,000	0.3	0.111	0.107
1.0	1000	1.0	1.0	1,000	1.0	0.231	0.122
1.0	1000	0.3	0.3	1,000	0.3	0.3	0.372
1.0	100	1.0	1.0	100	1.0	0.752	0.124
1.0	$+\infty$[a]	0.1	0.1	27,796	0.1	0.1	0.114

[a] This setting is equivalent to executing IGD algorithm.

In this set of experiments, we evaluated the most severe situation by assuming m as 1.0. We changed the budget requirements, including the max credit threshold (T_{max}), the max replication probability (r_{max}) and the max verification probability (v_{max}), and observed the TIGD suggested parameters and their corresponding optimal job error rates. In the experiments, we set other parameters as fixed values listed in Table 2.

The experimental results are shown in Table 3. Table 3 shows that in each system setting, the TIGD algorithm can find the optimal job error rate under the user's budget requirements. For instance, when T_{max} was 10,000, r_{max} and v_{max} were 0.3, the TIGD algorithm can find an optimal job error rate as 0.107%, with T_{opt} as 10,000, r_{opt} as 0.3 and v_{opt} as 0.111. When T_{max} was 1000, r_{max} and v_{max} were 1.0, the TIGD algorithm can find an optimal job error rate as 0.122%, with T_{opt} as 1000, r_{opt} as 1.0 and v_{opt} as 0.231. When T_{max} was 1000, r_{max} and v_{max} were 0.3, the optimal value J_{opt} that the algorithm can find is 0.372%. Although such a job error rate was high, it was the best value that IAOC can achieve under user budget requirements. In order to further reduce the job error rate, the user had to either increase T_{max}, r_{max} or v_{max}. The fourth row in the table indicates that when T_{max} was small (such as 100), the TIGD algorithm can still achieve a small value of J_{opt} (0.124%), if r_{opt} and v_{opt} were set to a large value.

The configuration of the last row of the table was equivalent to execution of an IGD algorithm. It sets the value of r_{max} and v_{max} the same as their initial values ($r_{init} = 0.1$ and $v_{init} = 0.1$, respectively). T_{max} was set to infinity, which means that T can be increased unboundedly. This experiment was used to compare with the TIGD algorithm. The first row of this table had the same initial system configure and generated the same job error rate. We notice that, in order to achieve the optimal job error rate 0.114%, T_{opt} has to be set to 27,796 in the IGD algorithm. Compared to the first row of Table 3, which achieved the similar job error rate,

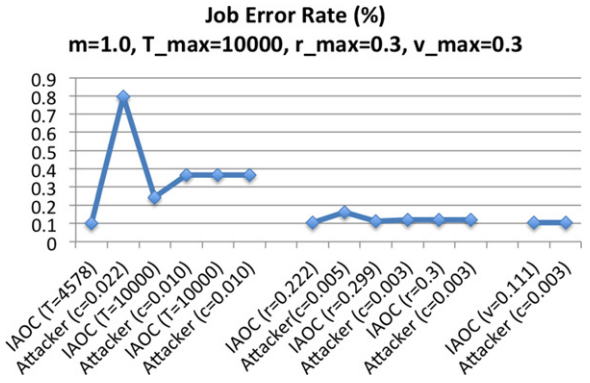

Fig. 3. Tiered interactive gradient descent execution detail.

Fig. 4. Interactive gradient descent execution detail.

the resulting T_{opt} that was generated by IGD algorithm was 178% higher than the one generated by TIGD algorithm. The result shows that in order to achieve a small job error rate, raising r_{max} and v_{max} slightly (to 0.3) can significantly reduce the value of the credit threshold T_{opt}.

To give readers an intuitive understanding of the working procedure of TIGD and IGD algorithm, we recorded the job error rate change during the algorithm executions. Fig. 3 shows the execution details of TIGD when setting parameters as the first row of Table 3. The three disjoint curves reflect three iterations of interactive gradient descent algorithm. In the first iteration of interactive gradient descent, IAOC and the attacker take turn to adjust T and c to minimize their loss function values. We observe that after three rounds of interactions, IAOC achieves the upper bound of T (10,000). The attacker finds the optimal value of c that maximize its loss function when IAOC set T to 10,000. (Notice that even though in the figure the second and third round appears to be the same value of T and c, their values are actually different with a higher number precision.) The second curve shows the second iteration of interactive gradient descent, where IAOC and the attacker adjust r and c respectively to minimize their loss function values. Notice that adjusting r has a significant effect in helping reducing job error rate. After the first round of interactive gradient descent, the job error rate is 0.364%. After the second round of interactive gradient descent, the job error rate drops to 0.119%. In the last iteration of interactive gradient descent, IAOC and the attacker adjust the value of v and c respectively to minimize their loss function values. After this iteration, the job error rate finally drops to 0.107%. The algorithm returns optimal system parameters: T_{opt} as 10,000, r_{opt} as 0.3 and v_{opt} as 0.111.

Fig. 4 shows the execution details of IGD when setting parameters as the last row of Table 3. We observe that after four rounds of interactive gradient descent, the algorithm can also achieve low job error rate (0.114%). However, it requires a higher value of credit threshold (i.e. $T_{opt} = 27{,}796$).

7. Related work

Task scheduling with budget restriction. Most of solutions toward budget restricted task scheduling falls into one of two directions: formal and heuristics. In formal method, the system is usually modeled as a linear programming problem [7–10], which needs to achieve a certain goal (e.g., maximize the execution performance) within a set of constraints (e.g., job execution cost or deadlines). By solving linear programming equation, the solution suggests a set of parameters, which achieves the goal, while satisfying the constraints. In heuristic method [11,12], the solution is to break the scheduling plan into multiple steps, which are usually sorted by priority. During the scheduling plan making, the solution finds optimal solutions step by step with greedy strategy. The solution ensures the requirement with a high priority is firstly satisfied.

Most budget restricted task scheduling solutions assume tasks are executed in a trusted environment, and thus does not consider the existence of malicious workers. In our work, we treat the result integrity as the first-class citizen, offer a solution that can achieve high accuracy even if the malicious worker is able to choose the best strategy to sabotage the computation.

Integrity assurance of outsourced computing. Researchers have studied such a problem for decades and proposed a wide range of solutions.

Secure co-processor [13] provides a hardware-based solution toward the trusted outsourced computing. The secure hardware chip Trusted Platform Modules (TPMs) installed on computers can verify the integrity of the worker's software stack ranging from the bios configuration at the booting procedure to the application that executes the outsourced task. By verifying the computing system completely complies the software stack specification, such a technology can guarantee the correctness of task result [14]. However, the solution requires the configuration of the worker's machine to be exactly identical to the TPM's expectation. Such a strict requirement is only suitable for the dedicated worker, which is used for certain computation task. But it is not suitable for the voluntary computing or P2P computing [2,3]. Further more, such a solution is useless on crowdsourcing based outsourced computing.

Another class of solution is using zero-knowledge proof [15–17]. It requires interaction between the master (verifier) the worker (prover). If the worker can submit proof to the master and the master verifies the proof, the master believes the assigned task is faithfully executed and accept the result. Gennaro et al. [18] propose a zero-knowledge protocol that requires no interaction. Although such a class of solutions is promising, they are only efficient on certain special applications, therefore are not suitable for generalized outsourced computing.

A class of practical solutions toward such a problem falls to the direction of task replication and verification. Our solution is following this direction.

The byzantine fault tolerance algorithm [19–25] leverages redundancy to address such a problem. However, the high dependency of such a solution is built upon the cost of high redundancy, which makes it impractical in outsourced computing environment. Theoretical study in [26] shows that to guarantee the storage (or computation) integrity, it takes at least $3f + 1$ replicas, where f is the number of faulty (malicious) storage nodes (or computation workers). For the outsourced computing, that means the number of replicas for each task is at least four (including the original task). Our method, which leverages probabilistic task replication and probabilistic task verification, can reduce most tasks' replica numbers to at most two (some tasks might be replicated more than once due to the task reschedule). In addition, the byzantine fault tolerance algorithm works in the environment where malicious worker fraction is less than 1/3, i.e., $m < 1/3$,

whereas our solution can work even if malicious workers dominate the environment. The limitation of our solution is that we can only guarantee a high portion of tasks to be computed correctly, i.e., $J < 1.0$, where the byzantine fault tolerance algorithm can guarantee complete integrity.

Other researchers propose solutions with partial redundancy and partial verification. Golle et al. [27] propose to duplicate computation tasks in distributed computation environment, and make theoretical study on the replication probabilistic distribution that forces the malicious worker to cooperate, rather than defect. However, similar to the byzantine fault tolerance algorithm, such a scheme requires a small or even a negligible fraction of malicious workers in the outsourced environment. Zhao et al. [28] propose to insert indistinguishable *quizzes* to the task package and assign quizzes along with regular tasks to workers. The master knows the results of those quizzes, and thus is able to verify the quiz answers quickly and detect malicious workers. Their simulation result shows that by combining reputation system, quiz approach gains a higher accuracy and a lower overhead than the replication-based approach. However, suggested by their simulation, the reputation accumulation is a long-term process so that in order to accumulate a reliable reputation, it takes as many as 10^5 tasks to execute for a worker. In addition, this solution does not consider the intelligent malicious worker who behaves honestly at the beginning to earn a reliable reputation, and cheats afterwards.

Some researchers propose further improvement solutions to the original task replication and verification idea, e.g., [29–31]. While others apply the idea on specific applications. For example, [32] propose a variant of task verification solution targeting on one-way hash function computation. [33,34,4] and [5] are targeting on MapReduce applications. [4] and [5] combine the task replication, task verification and reputation system techniques, propose and implement an integrity assurance MapReduce system on a hybrid cloud architecture.

This paper extends the solution in [4,5] to the generalized outsourced computing problem. In addition, we perform further theoretical analysis, and propose two algorithms that can find the optimal system parameter setting within user's accuracy requirement without and with considering user's budget.

8. Conclusion

Protecting the correctness of outsourced computing is a critical and challenging problem. In this paper, we extended the existing solution in [4] and [5] to propose Integrity Assurance Outsourced Computing (IAOC) system, which offers high accuracy guarantee for generalized outsourced computing. Based on the IAOC system, we built a model to compute the job error rate, a metric of job result accuracy. Based on the theoretical analysis, we proposed the Interactive Gradient Descent (IGD) algorithm and the Tiered Interactive Gradient Descent (TIGD) algorithm that can search for the optimal parameter setting to satisfy user's accuracy requirement, without or with considering user's budget requirements. We proved that the IGD and TIGD algorithm will find the optimal parameters that guarantee the job error rate to be the lowest upper bound. Our experiments showed that even in the most severe situation, where the malicious nodes completely dominate the outsourced computing environment, IGD and TIGD algorithm still can find the parameter setting that satisfies user's budget and accuracy requirements.

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.

Appendix. Proof of Theorem 4.1

Proof. S_n, the probability of a malicious worker to survive after executing n original tasks is the probability summation of all permutations on n independent events. Each event should fall into one of the below three cases:

(1) The worker does not cheat. The probability in this case is $(1-c)$.
(2) The worker cheats, but the task is not replicated. The probability in this case is $c(1-r)$.
(3) The worker cheats, and the task is replicated. However, the other worker executing the replicated task can collude. When the consistent results pass the first-layer check, the task result is not verified. The probability in this case is $crm(1-v)$.

By summing up the probability of difference permutations of above three cases on n independent events, we have

$$S_n = \sum_{i=0}^{n}\sum_{j=0}^{n-i}\left(\binom{n}{i}\binom{n-i}{j}(1-c)^i(c(1-r))^j(crm(1-v))^{n-i-j}\right).$$

By applying multinomial theorem, we have

$$S_n = (1-cr+crm(1-v))^n.$$

We are now deriving the value of J. Suppose the credit threshold is T, the probability that the master accept exactly k out of T incorrect task results consists of $(T-k)$ events that the worker does not cheat, and k events that the worker cheats but undetected. That is

$$\Delta_k = \sum_{i=0}^{k}\left(\binom{T}{T-k}\binom{k}{i}(1-c)^{T-k}(c(1-r))^i(crm(1-v))^{k-i}\right).$$

The expected number of tasks returning incorrect results in a batch (i.e. in T tasks) is

$$\begin{aligned}E &= \sum_{k=0}^{T}(k\cdot\Delta_k)\\ &= \sum_{k=0}^{T}\left(k\cdot\sum_{i=0}^{k}\left(\binom{T}{T-k}\binom{k}{i}\right.\right.\\ &\quad\left.\left.\times(1-c)^{T-k}(c(1-r))^i(crm(1-v))^{k-i}\right)\right).\end{aligned}$$

The error rate of T tasks is therefore

$$e = \frac{E}{T}.$$

Since the task assignment is uniformly distributed on all workers, and the malicious worker fraction m stays constant. We have the error rate in a job (i.e., job error rate):

$$\begin{aligned}J &= m\cdot e + (1-m)\cdot 0\\ &= me\\ &= \frac{m}{T}\cdot\sum_{k=0}^{T}\left(k\cdot\sum_{i=0}^{k}\left(\binom{T}{T-k}\binom{k}{i}\right.\right.\\ &\quad\left.\left.\times\ (1-c)^{T-k}(c(1-r))^i(crm(1-v))^{k-i}\right)\right).\end{aligned}$$

We are now simplifying the equation J. By reorganizing equation J, we have

$$\begin{aligned}J = \frac{m}{T}\cdot\sum_{k=0}^{T}&\left(k\cdot\binom{T}{T-k}\cdot(1-c)^{T-k}\right.\\ &\left.\cdot\sum_{i=0}^{k}\left(\binom{k}{i}(c(1-r))^i(crm(1-v))^{k-i}\right)\right).\end{aligned}$$

By applying multinomial theorem, to the last summation term, we have

$$J = \frac{m}{T} \cdot \sum_{k=0}^{T} \left(k \cdot \binom{T}{T-k} \cdot (1-c)^{T-k} \cdot \Big(c(1-r) + crm(1-v) \Big)^{k} \right).$$

For simplicity, we define

$$A = 1 - c$$

$$B = crm(1-v).$$

By replacing k with $T - l$, we have

$$J = \frac{m}{T} \cdot \sum_{l=0}^{T} \left((T-l) \cdot \binom{T}{l} \cdot A^{l} \cdot B^{T-l} \right).$$

Expanding $\binom{T}{l}$, we have

$$\begin{aligned} J &= m \cdot \sum_{l=0}^{T} \frac{(T-1)!}{l! \cdot (T-l-1)!} \cdot A^{l} \cdot B^{T-l} \\ &= m \cdot \sum_{l=0}^{T} \binom{T-1}{l} \cdot A^{l} \cdot B^{T-l} \\ &= m \cdot \left(\sum_{l=0}^{T-1} \binom{T-1}{l} \cdot A^{l} \cdot B^{T-l} + \binom{T-1}{T} \cdot A^{T} \right). \end{aligned}$$

By binomial definition, we have $\binom{T-1}{T} = \frac{(T-1)!}{T! \cdot (-1)!}$. By factorial definition $n! = (n+1)!/(n+1)$, we have $0! = 1!/1 = 1$ and $(-1)! = 0!/0 = 1/0$. By replacing $\binom{T-1}{T}$ with factorial form and replacing $(-1)!$ with $1/0$, we have

$$\begin{aligned} J &= m \cdot \left(\sum_{l=0}^{T-1} \binom{T-1}{l} \cdot A^{l} \cdot B^{T-l} + \frac{(T-1)!}{T! \cdot (-1)!} \cdot A^{T} \right) \\ &= m \cdot \left(\sum_{l=0}^{T-1} \binom{T-1}{l} \cdot A^{l} \cdot B^{T-l} + \frac{1}{T \cdot (1/0)} \cdot A^{T} \right) \\ &= m \cdot \Big(B \cdot (A+B)^{T-1} + 0 \Big) \\ &= m \Big(c(1-r) + crm(1-v) \Big) \Big(1 - cr + crm(1-v) \Big)^{T-1}. \end{aligned}$$

References

[1] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113. http://dx.doi.org/10.1145/1327452.1327492. URL http://doi.acm.org/10.1145/1327452.1327492.

[2] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, Seti@home: An experiment in public-resource computing, Commun. ACM 45 (11) (2002) 56–61. http://dx.doi.org/10.1145/581571.581573. URL http://doi.acm.org/10.1145/581571.581573.

[3] D. Anderson, Boinc: a system for public-resource computing and storage, in: Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on, 2004, pp. 4–10. http://dx.doi.org/doi:10.1109/GRID.2004.14.

[4] Y. Wang, J. Wei, M. Srivatsa, Result integrity check for mapreduce computation on hybrid clouds, in: IEEE CLOUD, 2013.

[5] Y. Wang, J. Wei, M. Srivatsa, Cross cloud mapreduce: A result integrity check framework on hybrid clouds, Int. J. Cloud Comput. 1 (1) (2013) 26–39.

[6] J. Douceur, The sybil attack, in: P. Druschel, F. Kaashoek, A. Rowstron (Eds.), Peer-to-Peer Systems, in: Lecture Notes in Computer Science, vol. 2429, Springer, Berlin, Heidelberg, 2002, pp. 251–260. http://dx.doi.org/10.1007/3-540-45748-8_24.

[7] Y.C. Lee, C. Wang, A.Y. Zomaya, B.B. Zhou, Profit-driven service request scheduling in clouds, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGRID'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 15–24. http://dx.doi.org/10.1109/CCGRID.2010.83.

[8] R. Van den Bossche, K. Vanmechelen, J. Broeckhove, Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads, in: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, 2010, pp. 228–235. http://dx.doi.org/10.1109/CLOUD.2010.58.

[9] M. Mao, J. Li, M. Humphrey, Cloud auto-scaling with deadline and budget constraints, in: Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on, 2010, pp. 41–48. http://dx.doi.org/10.1109/GRID.2010.5697966.

[10] A. Oprescu, T. Kielmann, Bag-of-tasks scheduling under budget constraints, in: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, 2010, pp. 351–359. http://dx.doi.org/10.1109/CloudCom.2010.32.

[11] M. Xu, L. Cui, H. Wang, Y. Bi, A multiple qos constrained scheduling strategy of multiple workflows for cloud computing, in: Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on, 2009, pp. 629–634. http://dx.doi.org/10.1109/ISPA.2009.95.

[12] M. Mao, M. Humphrey, Scaling and scheduling to maximize application performance within budget constraints in cloud workflows, in: Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, 2013, pp. 67–78. http://dx.doi.org/10.1109/IPDPS.2013.61.

[13] R. Sailer, X. Zhang, T. Jaeger, L. van Doorn, Design and implementation of a tcg-based integrity measurement architecture, in: Proceedings of the 13th Conference on USENIX Security Symposium - vol. 13, SSYM'04, USENIX Association, Berkeley, CA, USA, 2004, pp. 223–238. URL http://dl.acm.org/citation.cfm?id=1251375.1251391.

[14] A. Ruan, A. Martin, Tmr: Towards a trusted mapreduce infrastructure, in: Services (SERVICES), 2012 IEEE Eighth World Congress on, 2012, pp. 141–148. http://dx.doi.org/10.1109/SERVICES.2012.28.

[15] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof systems, SIAM J. Comput. 18 (1) (1989) 186–208. http://dx.doi.org/10.1137/0218012.

[16] O. Goldreich, S. Micali, A. Wigderson, Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems, J. ACM 38 (3) (1991) 690–728. http://dx.doi.org/10.1145/116825.116852. URL http://doi.acm.org/10.1145/116825.116852.

[17] M. Bellare, O. Goldreich, On defining proofs of knowledge, in: E. Brickell (Ed.), Advances in Cryptology CRYPTO 92, in: Lecture Notes in Computer Science, vol. 740, Springer, Berlin, Heidelberg, 1993, pp. 390–420. http://dx.doi.org/10.1007/3-540-48071-4_28.

[18] R. Gennaro, C. Gentry, B. Parno, Non-interactive verifiable computing: Outsourcing computation to untrusted workers, in: T. Rabin (Ed.), Advances in Cryptology CRYPTO 2010, in: Lecture Notes in Computer Science, vol. 6223, Springer, Berlin, Heidelberg, 2010, pp. 465–482. http://dx.doi.org/10.1007/978-3-642-14623-7_25.

[19] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401. http://dx.doi.org/10.1145/357172.357176. URL http://doi.acm.org/10.1145/357172.357176.

[20] M. Castro, B. Liskov, et al., Practical byzantine fault tolerance, in: OSDI, vol. 99, 1999, pp. 173–186.

[21] M. Castro, B. Liskov, Practical byzantine fault tolerance and proactive recovery, ACM Trans. Comput. Syst. 20 (4) (2002) 398–461. http://dx.doi.org/10.1145/571637.571640. URL http://doi.acm.org/10.1145/571637.571640.

[22] M. Abd-El-Malek, G.R. Ganger, G.R. Goodson, M.K. Reiter, J.J. Wylie, Fault-scalable byzantine fault-tolerant services, in: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles, SOSP'05, ACM, New York, NY, USA, 2005, pp. 59–74. http://dx.doi.org/10.1145/1095810.1095817. URL http://doi.acm.org/10.1145/1095810.1095817.

[23] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, L. Shrira, Hq replication: A hybrid quorum protocol for byzantine fault tolerance, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI'06, USENIX Association, Berkeley, CA, USA, 2006, pp. 177–190. URL http://dl.acm.org/citation.cfm?id=1298455.1298473.

[24] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, E. Wong, Zyzzyva: Speculative byzantine fault tolerance, in: Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles, SOSP'07, ACM, New York, NY, USA, 2007, pp. 45–58. http://dx.doi.org/10.1145/1294261.1294267. URL http://doi.acm.org/10.1145/1294261.1294267.

[25] P.-L. Aublin, S. Ben Mokhtar, V. Quema, Rbft: Redundant byzantine fault tolerance, in: Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on, 2013, pp. 297–306. http://dx.doi.org/10.1109/ICDCS.2013.53.

[26] M. Pease, R. Shostak, L. Lamport, Reaching agreement in the presence of faults, J. ACM 27 (2) (1980) 228–234. http://dx.doi.org/10.1145/322186.322188. URL http://doi.acm.org/10.1145/322186.322188.

[27] P. Golle, S. Stubblebine, Secure distributed computing in a commercial environment, in: P. Syverson (Ed.), Financial Cryptography, in: Lecture Notes in Computer Science, vol. 2339, Springer, Berlin, Heidelberg, 2002, pp. 289–304. http://dx.doi.org/10.1007/3-540-46088-8_23.

[28] S. Zhao, V. Lo, C. Dickey, Result verification and trust-based scheduling in peer-to-peer grids, in: Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on, 2005, pp. 31–38. http://dx.doi.org/10.1109/P2P.2005.32.
[29] W. Du, J. Jia, M. Mangal, M. Murugesan, Uncheatable grid computing, in: Distributed Computing Systems, 2004. Proceedings. 24th International Conference on, 2004, pp. 4–11. http://dx.doi.org/10.1109/ICDCS.2004.1281562.
[30] D. Szajda, B. Lawson, J. Owen, Toward an optimal redundancy strategy for distributed computations, in: Cluster Computing, 2005. IEEE International, 2005, pp. 1–11. http://dx.doi.org/10.1109/CLUSTR.2005.347045.
[31] D. Szajda, B. Lawson, J. Owen, Hardening functions for large scale distributed computations, in: Security and Privacy, 2003. Proceedings. 2003 Symposium on, 2003, pp. 216–224. http://dx.doi.org/10.1109/SECPRI.2003.1199338.
[32] P. Golle, I. Mironov, Uncheatable distributed computations, in: D. Naccache (Ed.), Topics in Cryptology CT-RSA 2001, in: Lecture Notes in Computer Science, vol. 2020, Springer, Berlin, Heidelberg, 2001, pp. 425–440. http://dx.doi.org/10.1007/3-540-45353-9_31.
[33] W. Wei, J. Du, T. Yu, X. Gu, Securemr: A service integrity assurance framework for mapreduce, in: Proceedings of the 2009 Annual Computer Security Applications Conference, ACSAC'09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 73–82. http://dx.doi.org/10.1109/ACSAC.2009.17.
[34] Y. Wang, J. Wei, Viaf: Verification-based integrity assurance framework for mapreduce, in: IEEE CLOUD, 2011, pp. 300–307.

Yongzhi Wang received his B.S. and M.S. degrees in Computer Science from Xidian University, China in 2004 and 2007, respectively. He received his Ph.D. in Computer Science from Florida International University, FL, USA in 2015. Since August 2015, he has been an Assistant Professor at Xidian University, China. His research interests include big data, cloud computing security and outsourced computing security.

Jinpeng Wei received a Ph.D. in Computer Science from Georgia Institute of Technology, Atlanta, GA in 2009. He is currently an Assistant Professor at the School of Computing and Information Sciences, Florida International University, Miami, FL. His research interests include malware detection and analysis, information flow security, cloud computing security, and file-based race condition vulnerabilities. He is a member of the IEEE and the ACM.

Shaolei Ren received his B.E., M.Phil. and Ph.D. degrees, all in Electrical Engineering, from Tsinghua University in 2006, Hong Kong University of Science and Technology in 2008, and University of California, Los Angeles, in 2012, respectively. From 2012 to 2015, he was with Florida International University as an Assistant Professor. Since July 2015, he has been an Assistant Professor at University of California, Riverside. His research interests include power-aware computing, data center resource management, and network economics.

Yulong Shen received the B.S. and M.S. degrees in Computer Science and Ph.D. degree in Cryptography from Xidian University, China, in 2002, 2005, and 2008, respectively. He is currently a Professor at the School of Computer Science and Technology, Xidian University, China. He is also the associate director of the Shaanxi Key Laboratory of Network and System Security and a member of the State Key Laboratory of Integrated Services networks Xidian University, China. He serves as a general co-chair or a technical program committee member for several international conferences, including ICEBE, INCoS, NANA, CIS and SOWN. His research interests include wireless network security and cloud computing security.