Optimized Replication Management with Reputation for Detecting Collusion in Large Scale Cloud Systems

Hanane Bennasar
ENSIAS Laboratory, ENSIAS
Mohamed V University

Mohammad Essaaidi
ENSIAS Laboratory, Mohamed V University, Rabat, Morocco

Ahmed Bendahmane
ITS Laboratory, Faculty of Science
Abdelmalek Essaadi University, Tetouan, Morocco

Jalel Ben-othman
L2ti Laboratory, Paris13 Villetaneuse University
Paris, France

ABSTRACT
Several cloud computing systems used voting techniques to deal with sabotage issues. However, these techniques become inefficient, and present some new security vulnerabilities when malicious resources collude and return the same wrong result. Usually, this kind of security threats are handled using several techniques and approaches such as voting techniques. In this paper, a very efficient approach to overcome sabotage issues is proposed, especially in the case of very complex attacks. The performances of this approach are evaluated in a cloud system model and it is compared against other voting techniques, like reputation-based voting, using simulations which allowed to investigate the effect of collusive cloud resources on the correctness of the results. The obtained results show that the proposed approach achieves lower error rates and enhanced performances in terms of overhead and slowdown.

General Terms
Cyber-security.

Keywords
Cloud computing; Cloud model; Voting; Collusion Attacks

1. INTRODUCTION
Cloud Computing is a large-scale distributed computing system which has initially emerged from financial systems. It consists of a pool of virtualized dynamically scalable computing power and storage platforms using virtualization strategies. The fact that cloud computing is based on distributed systems makes it vulnerable to several security and privacy threats [1]. Among the most important cloud computing security challenges there are Security related to Third Party Resources, Application Security, Data Transmission Security and Data Storage Security. Among the many Cloud Computing platforms that can be considered, this paper focuses on the Cloud system model.

The primary issue that the cloud framework experiences is result sabotage in the presence of malicious resources, particularly when they collude together and restores the equivalent flawed outcome. To overcome this issue, several mechanisms can be used such as reputation with replication procedures which are the basis of the proposed approach in this paper.

Replication [2] is utilized for expanding availability and enhancing execution of the system. Job replications are typically utilized for sabotaging tolerance to manage accuracy confirmation of job results in many cloud frameworks.

High-availability is among the main requirements of a cloud. This means anywhere and anytime access to services, tools and data. However, availability is one of the very few performance parameters that are part of the Service Level Agreements (SLAs) of today's cloud providers. The used method allowed the computation of resources availability to optimize their use. Majority voting and *m*-first voting [3] replicate a job to many independent resources and the returned results are verified for most of the decision. These techniques reduce the performance of cloud systems because they are very expensive in terms of resource utilization. To deal with this issue, other complex voting-based techniques using spot-checking as in credibility-based voting or combined with reputation system. However, these techniques rely on the assumption that the cloud resources behave independently. These techniques [30] are useless where several collusive resources collectively return the same wrong results of a job execution. In fact, in distributed cloud systems, a group of opponents may present some form of collective misbehaviour. Recently, several approaches were introduced to deal with collusion issues. A**.** Bendahmane et al. [4] proposed the Reputation-based Voting (RBV) scheme that enhances the credibility-based voting to solve the sabotage of computing resources.

The reputation mechanism [5] represents a significant technique for distributed resources behavior evaluation based on previous practices, and for enhancing reliability. As a major aspect of security instruments, reputation techniques have been proposed to improve reliability assessment of different entities in distributed computing.

In [6], if computing resources are also malicious, a scheduler does not need result certification only to guarantee the rightness of results. It uses reputation as well in order to select trustworthy resources or to eliminate malicious resources. The basic idea of reputation is to select for each node a trust value based on its behavior history and save that value properly in the system.

The proposed approach is named "an optimized replication management with reputation approach" (ORMR). The main idea behind ORMR is to optimize the use of replicas. In distributed systems, the important workload of jobs, computations, etc., limits the advantage of using replication especially when the number of replicas is used randomly. Moreover, if the number of replicas surpasses the best threshold, the unused replica would produce an overhead due to added messages communication.

To achieve high scalability and low overhead, job replication is reduced, consistency checking without using spot checkers is verified, and reputation can for the most part be formulated as high parallelized cloud system computations. Thus, security levels offered to the cloud scales up with its computational power.

To this end, the following approach is proposed:

- An intra-cloud reputation and replication system for the proposed cloud architecture is implemented and evaluated.
- This system approves a data-centric approach that identifies job and task replica deviations (the randomly use of job and task replicas).
- The use of replicas is not arbitrary; it depends on the max of reputation value.
- The use of dynamic blacklisting has been shown to optimize the blacklisting error metric.
- A distributed computing model illustrates how it can attain a scalable certification and reputation-tracking in the cloud.

The remainder of this paper is organized as follows. In section 2 a background and related works are provided, Section 3 is dedicated to illustrating the system model. In Section 4, the proposed approach for collusion tolerance is described. Section 5 presents the performance evaluation of the proposed method using Cloudsim simulations. Section 6 presents the main conclusions.

2. BACKGROUD AND RELATED WORKS

2.1 Sabotage Tolerance Mechanisms

To eliminate the effect of fault results, a combination of sabotage tolerance techniques must be included in the system. The most popular approaches used for sabotage tolerance are the voting techniques and replication.

2.1.1 M-First Voting

It's the specific fundamental utilized system for large scale distributed computing especially when malicious resources do not communicate with each other [7]. In the M-first voting technique, a master replicates a job and designates them to a group of workers. The result which collects m matching values first is accepted as the final result for the job. Specifically, when m is set to a minimum value, it has a minimum redundancy, so the performance of the system may be essentially decreased.

2.2.1 SPOT-CHECKING

In spot-checking [8], a master sometimes assigns spotter jobs whose correct results are already known to the master. With this technique, the master can directly check if workers behave maliciously or not. If a worker [29] returns an incorrect result, the master considers the worker as a saboteur. In that case, the master may use the backtracking policy, so it can countermeasure against the saboteur, so all results returned from the saboteur are cancelled, as well as the master can use the blacklisting technique by submitting wrong results is prevented.

2.3.1 CREDIBILITY BASED VOTING

It combines the properties of the two techniques m-first voting and spot-checking, in order to achieve more reliability. In this method [9], a master accomplishes a weighted voting, so a master estimates the reliability of the workers based on their behaviors during the task. It is like m-first voting, but the difference is that the number of replications m is dynamically determined at the runtime in harmony with some credibility values given to different elements of the system: worker, result, result group, and task. These credibility values represent the reliability rates and are mainly based on the number of spot-checks given to workers and their past behavior.

To have the required capacity to check the credibility of workers, the master assigns spotters' task to a worker with probability p known as spot-check rate. Also, the master considers the worker as malicious when only one worker returns a result which does not match with the correct one. This result is received by the spotter task. For this reason, the master may use two techniques combined with spot-checking. Blacklisting or Backtracking. The master may use the backtracking to throw out all results received from the malicious worker. The master may use the blacklisting to take in identified malicious workers into a blacklist for preventing them to return results or taking more computation tasks.

The credibility-based voting (CBV) is an efficient voting technique which combines computations redundancy and reliability. However, CBV has two critical limitations: it leads to the dispersion of resources as in simple voting scheme, because it requires more computations to generate the result of a spotter jobs. It also suffers from the problems of malicious computing resources which can behave properly for a long period of time, by returning correct results of spotter jobs, in order to achieve high credibility and then start to sabotage the real jobs.

2.4.1 REPUTATION BASED VOTING

In this approach, A. Bendahmane et al. [4] improve the credibility-based voting technique using a spot checking technique. The basic idea of RBV is to check the computing resources without assigning spotter jobs and to consider the result of voting decision as the one of spot-checking to estimate the credibility without more computations. This credibility is considered as reputation which is used in the RBV decision. This reputation is used as a weight in the voting decision which is based on the weighted average voting method to improve the efficiency of replication-based voting techniques.

In RBV, spot-checking is used to check occasionally the computing resources. The spotter job is sent to those whose right outcome is known, in order to evaluate the credibility of each computing resource based on returned result.

In the RBV approach, blacklisting is used to compute a resource where results are not validated by RBV algorithm, and backtracking is applied before for all results returned by malicious resources.

As a result, the reputation Ri of any computing resource which returns the accepted result of ki task, when blacklisting is used, is computed using the following equation [3]:

$$R_i = CR(C_i, k_i) = \begin{cases} 1 - \dfrac{f}{1-f} \cdot \dfrac{1}{k_i e} & ,if\ k_i \neq 0 \\ 1 - f & ,otherwise \end{cases} \quad (1)$$

Where f is the proportion of malicious computing resources, e represents the base of the natural logarithm, and k is the number of times that w survives spot-checking.

The credibility of the worker C (w) is equal to the credibility of the result C(r):

$$C(r) = C(w) \quad (2)$$

The credibility of result group C(r) is the probability that all results in result group are correct.

Moreover, in order to check the trustworthiness of a result, RBV approach uses the m-first voting technique based on reputation decision. The resulting reputation R(Vj) of a given

result Vj is represented as the sum of the computing resources reputations' returning the result Vj.

$$R(Vj) = \sum_{i=1}^{n} T(Vj, Ci) Ri \quad (3)$$

Where T(Vj, Ci) is the relationship between the result Vj and the computing resource Ci, which is computed as follow.

$$T(V_j, C_i) = \begin{cases} 1 & ,if\ C_i\ compute V_j \\ 0 & ,otherwise \end{cases} \quad (5)$$

In the case of having only one replica of a task (i=j=1, in line 44 of table 1), the broker considers that the received result is correct if the reputation R1 of a computing resource C1 exceeds the minimum reputation (R1 > Rmin) (line 45 of table 1). Otherwise, the broker applies the reputation-based voting method for further replications and uses the reputation value of each computing resource to decide which result is accepted as correct. To this end, each time the broker receives a new result Vj for task replica, it recalculates the result reputation value R(Vj) (line 22 of table 1). Λ is denoted as the desired tolerance threshold ($0 < \lambda < 1$) and the result value is picked with the highest reputation max j(R(Vj)) as the best result (line 42 of table 1).

If $\max_{1 \le j \le m} (R(V_j)) > \lambda \sum_{i=1}^{n} R_i$ then the result value which represents this maximum is accepted by the broker and is considered the correct one (lines 49 and 50 of table 1). In addition, the reputation of all computing resources which generate this result is updated as follows (line 51):

$$R_i = CR(C_i, k_i + 1) \quad (2')$$

2.2 Related Works

During the last few years, the problem of collusion tolerance in large scale distributed computing has been addressed by many researches. Bendahmane et al. proposed [4] a voting method in order to deal with the collusion problem in large scale grid computing systems. Reputation based voting is an implementation of credibility-based voting and spot-checking. In RBV method, the behaviour of collusive computing resources is stable during the computation, each task is replicated n times and allocated to several computing resources.

The problem of collusion was also addressed by G. Levitin et al. [10] where the spot-checking optimization problem has been formulated and solved for grids subject to the collusive behavior. Also, an iterative method is proposed to evaluate the probability of genuine task failure (PGTF) and the expected overhead in terms of the total number of task assignments for the considered grid system.

Z. Zhu and R. Jiang [11] proposed a secure anti collusion data sharing scheme for dynamic groups in the cloud. They offer a secure way for key distribution without any secure communication channels, and the users can securely obtain their private keys from group manager. Also, the proposed scheme achieves fine-grained access control, any user in the group can use the source in the cloud and revoked users cannot access the cloud again. It is also proved that the scheme can achieve fine efficiency, which means previous users need not to update their private keys for the situation where either a new user joins the group, or a user is revoked from the group.

M. Mortazavi and B. Ladani [12] proposed A MapReduce-based algorithm for parallelizing collusion detection in Hadoop. A MapReduce-based algorithm for parallel collusion detection of malicious workers has been proposed. This algorithm utilizes a voting matrix that is represented as a list of voting values of different workers. Three phases of majority selection, correlation counting, and correlation computing are designed and implemented.

T. Samuel and A. Nizar [13] presented an efficient collusion-resistant method a credibility-based result verification scheme, which proceeds by running 'quiz' jobs on the slave nodes. Based on the results of quizzes and regular MapReduce jobs, the master node assigns credibility values to slave nodes, which are later used to verify the correctness of results produced by the nodes. The limitation of the approach is that, the result verification has to wait until the regular MapReduce jobs, which are long-running, complete their execution. This adversely affects the turn-around time, resource utilization and throughput of the system.

These limitations motivate K. Jiji and A.Nizar [14] to develop a new approach and to propose a new protocol called Intermediate Result Collection and Verification (IRCV) Protocol. This protocol employs quiz jobs and regular jobs to assign and update credibility of Worker Nodes by identifying malicious nodes early in the execution line. The protocol collects intermediate output for result verification and prunes out erroneous computation.

3. SYSTEM MODEL

2.3 System overview and overview and architecture

Herein a cloud system model consisting of one private cloud and one public cloud, is proposed. In the private cloud, a master node is deployed and a small number of slave nodes which are called verifiers. The workers are composed of the other slave nodes and Distributed File System, and they are deployed on a public cloud. The system model defines three types of tasks: the main task, the replication task, and the verification task. The main and replication tasks are executed by the workers on the public cloud. The verification tasks are executed by the verifier on the private cloud. The replication task validates the main tasks' result, because workers are not usually trusted. Moreover, the verification tasks ensure the replication task result by executing the task on the verifier, because the replication tasks are executed by the untrusted public cloud worker.

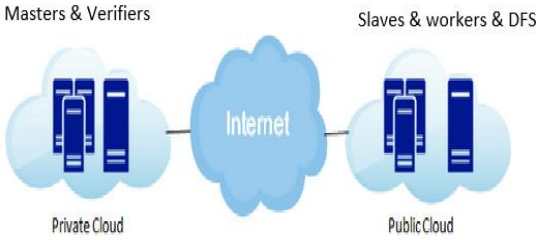

Fig 1. Architecture of proposed Cloud system model

3.2 Attacker model and assumptions

In our system model, the attack model represents several group of collusive computing resources distributed in diverse cloud service providers where malicious attackers launch security attacks in order to cooperate with the provided resources and exploit some of their weaknesses.

The multiple groups of collusive bad workers behave in coordination with other malicious workers and return the same erroneous result according to an agreement made between them, are assumed.

The verifiers and the DFS are trusted. But the master is not necessarily trusted. As a solution, it have been considered that the node that has the highest reputation value as a master.

In addition, it is supposed that the worker is not usually trusted, a good worker is honest and always returns the correct result for its task, but a bad worker may behave arbitrarily. If bad workers could tamper a good one, so this latter is considered as a bad worker too. Because, it should be able to protect data from being altered by unauthorized workers. Also, it have been assumed that the number of good workers is higher than that of the bad ones.

According to these assumptions, it have been supposed that the attack model composed of a whole of n bad malicious computing resources , also each malicious computing resource has a set of colluders, which is a subset of M denoted by S. In order to describe the attack model, it have been supposed that:

- Every malicious resource has a fix probability p ϵ [0, 1] to communicate with other colluders and to provide the same bad result.
- The probability that behaves correctly is then (1 – p).

4. THE PROPOSED APPROACH: OPTIMIZED REPLICATION MANAGEMENT WITH REPUTATION

The novel approach proposed, namely, Optimized Replication Management With Reputation (ORMR) is based on reputation and optimized replications. This approach improves the reputation-based voting (RBV) [5] by reducing the overhead and improving the accuracy. In reputation-based voting, each task is replicated n times and it is allocated to several computing resources Ci. However, a major limitation of this approach is inefficient use of resources and time to identify malicious resources.

The main idea of the approach is to reduce the use of replicas, by checking the list of computing resources and that of tasks. If the list of tasks is lower than the list of computing resources, the task is assigned to the computing resource which has a higher value of reputation max (R (Vj)).

Moreover, RBV blacklists all resources which return a reputation value below Rmin. In the approach, a dynamic blacklisting in the time is used. This is because a resource could be considered as a false malicious (Blacklisting error). Thus, a resource is blacklisted only a few minutes according to a parameter that is fixed during simulations, and it can be tested another time.

Table 1 represents the algorithm of our approach. Before the scheduling task, a task verification availability (line 11) is added, according to M. Haberkorn and K. Trivedi [15], MTTF and MTTR metrics are computed in order to verify the availability of resources, this allows to reduce the overhead with better accuracy.

Then in the tasks scheduling, the list of computing resources and the list of tasks are checked to optimize the use of replicas (line 23 of table 1). Since every replica must perform all updates eventually, there is a point beyond which adding more replicas does not increase the throughput, because every replica is saturated by applying updates. However, if the replication degree exceeds the optimal threshold, the useless replicas would generate an important overhead due to extra communication messages [16,31].

In the tasks related with result retrieval and validation the use of blacklisting (line 53 of table 1) is improved. However, in RBV, when a malicious resource is blacklisted, it is no longer used, and, thus, becomes useless. In ORMR approach, a blacklisted malicious resource could be trustworthy later. Therefore, it's proposed to add a time parameter, and to block the resource shortly, to use it later and to optimize the use of resources.

Table 1. ORMR Algorithm

1:	LT is the list of tasks to compute
2:	LC is the list of computing resources
3:	LCB is the list of computing resources blacklisted
4:	**Initialize mins= Timer()**
5:	**M=0 is a time parameter converted to minutes**
6:	$R_i = 1 - f$ and $k_i = 0$ for each $C_i \in$ LC, according to (1)
7:	**while** (there is task $T_k \in$ LT without accepted result) **do**
8:	Tasks scheduling ()
9:	Tasks result receiving and decision for acceptance ( )
10:	**end while**
11:	**Task verification of availability of computing resources ()**
12:	**for each** Ci in LC **do**
13:	**Compute** MTTF according to (8)
14:	**Compute** MTTR according to (7)
15:	**Compute** A according to (6)
16:	**if** Resource is available **then**
17:	RC_Status= "Available"
18:	**else**
19:	RC_Status="Unavailable"
20:	**end if**
21:	**end for**
22:	**Tasks scheduling ()**
23:	**while** (RC_STATUS = "Available") ∧ (there is task $T_k \in$ LT without accepted result) **do**
24:	**if** LT < LC **then**
25:	//Optimizing the use of replica
26:	Ci= **Fetch** Ci with most reputation value *max (R(V_j))*
27:	else
28:	*Ci= next computing resource in LC*
29:	**if** C_i is not blacklisted **then**
30:	T_r = a replica of T_k
31:	assign T_r to C_i
32:	**end if**

33: **end if**
34: **end while**
35: **Tasks result receiving and decision for acceptance ( )**
36: **while** (there is a running task) **do**
37: // waiting to receive a result back from a resource C_i
38: **for** each T_r executed for a task T_k **do**
39: receive a result V_j
40: **for** j = 1 to m **do**
41: compute $R(V_j)$ according to (1)
42: compute $max_j (R(V_j))$
43: **end for**
44: **if** a task T_k has only one replica with one result V_1 **then**
45: **if** $R(V_1) > R_{min}$ **then**
46: accept V_1
47: **end if**
48: **else**
49: **if** $max_j (R(V_j)) > \lambda \sum_{i=1}^{n} R_i$ **then**
50: accept V_j^* which represent this maximum
51: update R_i of all C_i^+ which generate V_j^*according to (2') and (1)
52: // add bad resources and mins to the blacklist
53: add all C_i^- and mins to the blacklist()
54: LCB = blacklist(C_i^-, mins)
55: **for** (r=0; r < list.lengh(); r++)
56: **if** (convert to minutes(mins – LCB(r) = > M) **then**
57: //remove blacklisted resource while M minutes
58: Remove (r)
59: **end if**
60: **end for**
61: // backtrack all the results returned by C_i^-
62: repeat from line **40** for all T_k that contain the backtracked results
63: **end if**
64: **end if**
65: **end for**
66: **end while**

To enhance the quality of computations, a new function called computing resources availability verification Task is added. in this step, the algorithm verifies the availability of resources after the use of computation tasks. To calculate a resource availability, the equation given below is used [17]:

$$Availability = \frac{MTTF}{MTTF + MTTR} \quad (6)$$

Where

$$MTTR = \frac{Da}{n} \quad (7)$$

$$MTTF = \frac{Ua}{m} \quad (8)$$

D_A is the aggregated downtime; where n is the number of downtime intervals.

U_A is the aggregated uptime; where m is the number of uptime intervals.

Haberkorn and Trivedi [15] proposed a quantified availability assurance, based on several approaches. In general, improving availability means increasing time-to-failure (TTF) and reducing time-to-recovery (TTR). To increase TTF, proactive failure avoidance techniques for aging-related bugs are used. To reduce TTR, it's proposed instead escalated levels of recovery, so that most failures are fixed by the quickest recovery method, and only few by the slowest ones.

5. PERFORMANCE EVALUATION

In order to evaluate the performance of the approach, a VO-system based cloud simulator, namely, Cloudsim [18] is used. Cloudsim is considered to be the most suitable simulator for the problem addressed in this paper. To this end, it have been developed some Cloudsim classes required for the implementation of ORMR approach. The aim of these simulations is to evaluate the efficiency of the proposed collusive resources tolerance approach and to compare the obtained results with respect to several performance metrics and parameters to the ones obtained with the reputation-based voting and credibility-based voting [19] [28]. Performance Metrics:

In order to assess the performance of our approach, several metrics have been considered, namely, the overhead [20], the slowdown [21], the Accuracy, the Error rate [22], and finally, the blacklisting error [23]:

Overhead = # A/B

Where: A = Total numbers of tasks assigned for execution

And B= The original number of tasks.

Slowdown= % of #running times of computations

Accuracy= #C/#D

Where: C= tasks with correct results & D= tasks accepted as correct.

Error rate=#E/#F

Where E= Number of the accepted erroneous task results and F= total number of the task results returned at the end.

Blacklisting error = #J/H

Where J=The number of non-collusive blacklisted resources and H= total of blacklisted resources.

5.1 Simulation Settings

As in the RBV approach, 10000 independent tasks have been considered in the simulations. The job is allocated to four cloud VOs, with each one consisting of 250 computing resources.

Also, it have been considered the following assumptions:

- All tasks have the same running time and all computing resources have the same computing power and can perform only one task at a time.
- All collusive resources can collude, with other resources in the same VO, and with collusive resources from other VOs, with the same probability c to return the same wrong result.
- For spot-checking rate, we set q to 0.1 as the best value according to Sarmenta method [9]. The parameter Rmin is seted to 0.995, which is assumed as a high value that must be exceeded by only one replica, in order to accept its result as correct.

Table 2. Simulation parameters

Symbol	Explanations	Values
f	Fraction of malicious collusion resources in cloud system	0 ~ 0.5
p	Probability that a collusive resource returns a wrong result	0 ~ 1
m	Redundancy	2
λ	Tolerance threshold	0.55 ~ 0.95
q	Spot-check rate	0.1

5.2 Results and discussion

5.2.1 Tolerance threshold

To compare our approach with other sabotage tolerance mechanisms, especially with RBV approach [4] [24] [25], Table 2 shows the simulations parameters.

Fig 1. allows the determination of the optimized tolerance value. When λ=0.75 our tolerance mechanism offers high accuracy, and the blacklisting error turns to zero for any specified values of f and p.

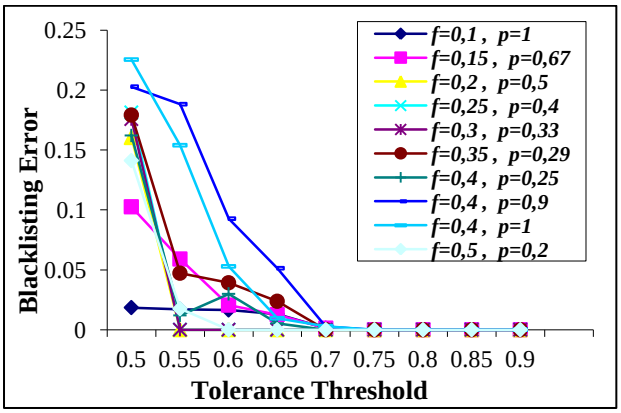

Fig 1. Blacklisting error as a function of tolerance threshold for different values of f and p

Fig 2. shows the blacklistig error as a function of tolerance threshold for several values of f and p. This metric decreases when λ increases. So, with dynamic blacklisting, the probability of blacklisting of non malicious resources is minimized. As a result, the blacklisting of collusive resources is becoming more efficient.

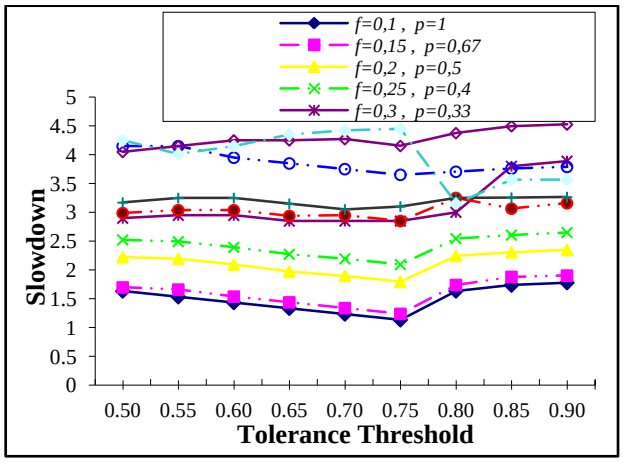

Fig 2. Overhead as a function of tolerance threshold for different values of f and p

Fig 3. shows the overhead as a function of the tolerance threshold, for λ ϵ [0.5,0.9]. The overhead increases slightly for all cases of f and c. In Figure. 4, the slowdown metric is also represented as a function of tolerance thresold, for λ ϵ [0.5,0.9]. The slowdown declines graduatly because of blacklisting, there are some blacklisted resources which are not used.

It have been noted that there is a significant increase of the overhead and decrease of the slowdown. Therefore, the blacklisting error curve suggests that λ=0.75 is the optimum value corresponding to a better efficiency.

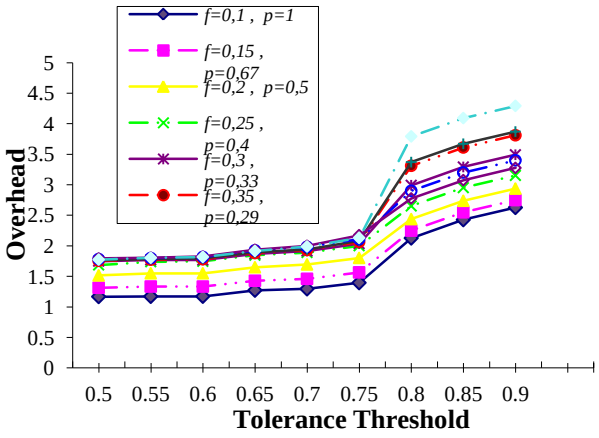

Fig 3. Slowdown as a function of tolerance threshold for different values of f and p

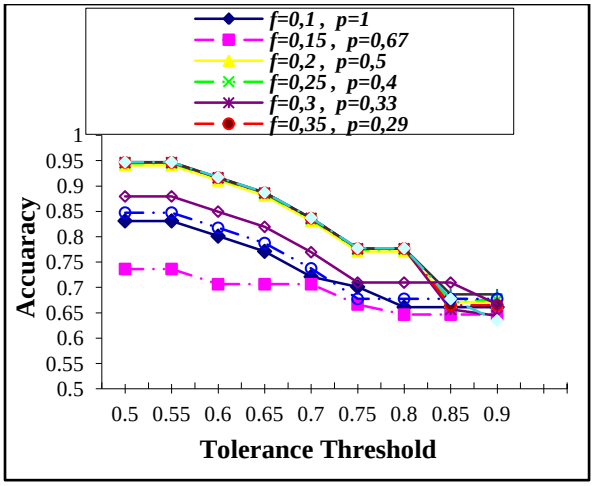

Fig 4. Accuracy as a function of tolerance threshold for different values of f and p

Fig 4. shows the accuaracy as a function of tolerance threshold for several values of f and p. Under collusion when the fraction of malicious resources increases and λ increases, the accuaracy declines graduatly. This is because computing resources need more time to finish the execution of all tasks launched.

5.2.2 Error rate and Overhead Variation

In Order to assess the performance of the proposed approach, the error rate and the overhead are computed [26] [27]. Then, the obtained results are compared with those obtained using reputation based voting, m-first voting and credibility-based voting.

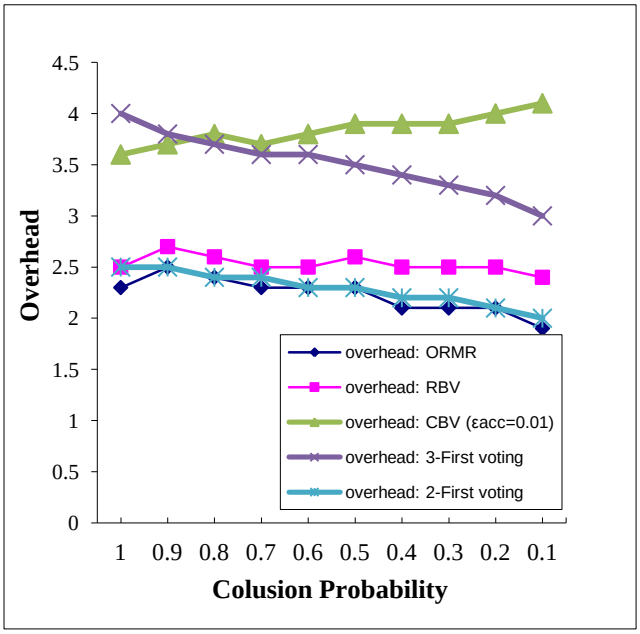

Fig 5. Overhead as a function of collusion probability

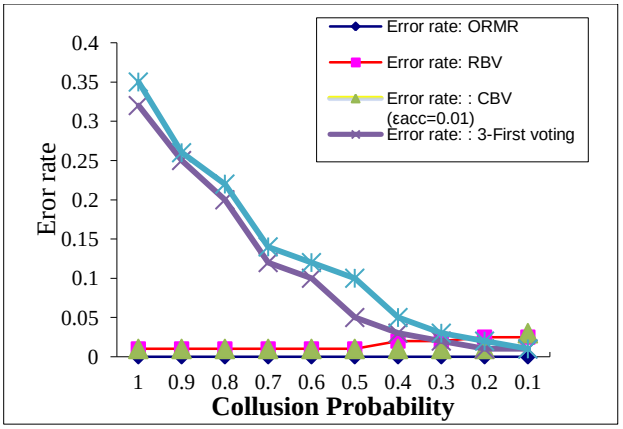

Fig 6. Error rate as a function of collusion probability

Fig. 5 shows that our method outperforms RBV approach and other methods considered. The overhead declines considerably, thanks to the dynamic Blacklisting and the optimization of the replicas' number (replicas are not used randomly). In ORMR approach, when many checks are executed through voting, computing resources can gain enough high reputation compared to RBV with the huge number of replicas. Then, the overhead for ORMR becomes smaller than RBV and other methods considered.

As far as the error rate is concerned, as illustrated in Fig. 6, in ORMR, RBV and CBV it tends to zero. However, it is noted that with a high collusion probability, the worst results are attained because the probability of reliable wrong results from collusive resources increases as the collusion probability increases.

These results mean that our approach provides much better performances compared to RBV, m-first voting and CBV approaches.

6. CONCLUSIONS

Security is usually listed as the number one concern for cloud computing adoption. Cloud security issues persistently rank above cloud reliability, network issues, availability and worries about the cloud financial profit.

We have presented in this paper an optimized approach for faulty tolerance sabotage system. This approach allowed to enhance RBV approach by using dynamic blacklisting, available resources and also by optimizing the number of replicas in order to achieve lower overhead and error rate with an acceptable accuracy. A low error rate and overhead were achieved against RBV, CBV and m-first voting approaches. In order to obtain high scalability, all trust management computations are formulated as distributed cloud computations. Therefore, our tolerance system for detecting collusive computing resources can be considered more accurate and more trustworthy.

As a future work, we propose to investigate the efficiency of our approach against a more complex attacks model based on a Hadoop system in the presence of collusive and malicious behaviors.

7. REFERENCES

[1] Z. Lei, C. Zuo-Ning, "MapReduce Implementation Based on vStarCloud", Journal of Advances in Computer Network, Vol. 1, No. 3, September 2013,pp 162-167.

[2] S.Zhao , V.Lo, "Result Verification and Trust-based Scheduling", Fifth IEEE International Conference on Peer-to peer Computing, IEEE Computer Society, Washington, pp. 31–38, 2005.

[3] C. Kumar, "Optimization of Data Storage and Fault Tolerance Strategies in Cloud Computing", International Journal of Science and Research (IJSR), 6.391 Volume 6, January 2017, pp 1320- 1324.

[4] A. Bendahmane, M. Essaaidi. "The Effectiveness of Reputation-based Voting for Collusion Tolerance in Large-Scale Grids", IEEE Transactions on Dependable and Secure omputing, Computing, 12, 6, (665), (2015).

[5] W.Jiang, H.Wan, S.Zhao, Reputation Concerns of Independent Directors: Evidence from Individual Director Voting, The Review of Financial Studies 29 (3), 655-696 , 2015.

[6] S.Choi, H.Kim, A Taxonomy of Desktop Grid Systems Focusing on Scheduling,Columbia Business school Reaserch 2015.

[7] K. Watanabe, M. Fukushi, and S. Horiguchi, "Collusion-Resistant Sabotage-Tolerance Mechanisms for Volunteer Computing Systems," IEEE International Conference on e-Business Engineering, Macau, pp. 213 –218, 2009.

[8] K. Watanabe, M.Fukushi, Optimal Spot-checking for Computation Time Minimization in Volunteer Computing,Springer, Graduate School of Information SciencesTohoku UniversityAoba-ku SendaiJapan, Grid Computing 7: 575.2009.

[9] L. F. G. Sarmenta, "Sabotage-tolerance mechanisms for volunteer computing systems,"Future Generation Computer Systems, 18(4 Future Gener. Comput. Syst. 18(4), 561–572 (2002) .

[10] G.Levitin, L.Xing,"Optimal Spot-Checking for Collusion Tolerance in Computer Grids", IEEE Transactions on Dependable and Secure Computing, doi:10.1109/TDSC.2017.2690293, 2017.

[11] Z.Zhu,R. Jiang, "A Secure Anti Collusion Data Sharing Scheme for Dynamic Groups in the Cloud", IEEE Transactions on Parallel and Distributed Systems, VOL. 27, NO. 1, JANUARY 2016.

[12] M. Mortazavi, B. Ladani A MapReduce-based algorithm for parallelizing collusion detection in Hadoop, IEEE International Conference, 145-154, 2015.

[13] T.Samuel, A.Nizar,"Credibility-Based Result Verification for Map-Reduce", India Conference (INDICON), 2014 Annual IEEE, pp. 1–6,2014.

[14] K. JIJI, A.Nizar, "An Efficient Approach for MapReduce Result Verification", Springer Science+Business Media Singapore, volume 412 2016.

[15] M.Haberkorn, K.Trivedi, "Availability Monitor for a Software Based System",IEEEXplore Conference: High Assurance Systems Engineering Symposium, 2007. HASE '07. 10th IEEE, 2007.

[16] M.Durrani ,A. Shamsi," Volunteer computing: requirements, challenges, and solutions", Journal of Network and Computer Applications archive Volume 39, March, 2014.

[17] M.Siebenhaar, O.Wenge, "Verifying the Availability of Cloud Applications", pp. 489-494, 2013.

[18] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya, CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software: Practice and Experience (SPE), Volume 41, Number 1, Pages: 23-50, ISSN: 0038-0644, Wiley Press, New York, USA, January, 2011.

[19] K.Watanabe, M.Fukushi," Adaptive group-based job scheduling for high performance and reliable volunteer computing". Journal of Information Processing, Volume 19, 2011.

[20] B. Ismail, D. Jagadisan, "Determining Overhead, Variance & Isolation Metrics in Virtualization for IaaS Cloud", Springer Science+Business Media, LLC , Pages 315-330, 2011.

[21] Kukanov, Alexey (2008-03-04). "Why a simple test can get parallel slowdown". Retrieved 2015-02-15

[22] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello, "Characterizing result errors in internet desktop grids," In Euro-Par 2007, Parallel Processing, 13th International Euro- Par Conference, France, Proceedings, volume 4641 of LNCS, pp. 361 371.Springer, 2007.

[23] M. Kuhn, S. Schmid, and R. Wattenhofer, "Distributed asymmetric verification in computational grids," in IEEE International Symposium on Parallel and Distributed Processing, p.1-10, 2008.

[24] A.Quamar, A.Deshpande, J.Lin, "NScale: Neighborhood-centric Large-Scale Graph Analytics in the Cloud", VLDB J., vol. 25, no. 2, pp. 125–150, 2016..

[25] J. Rittinghouse, J.Ransome, "Cloud computing: implementation, management, and security", CRC Press, Boca Raton, 2016.

[26] N.Bonvin, T.Papaioannou, K.Aber," A self-organized, fault-tolerant and scalable replication scheme for cloud storage". In Proc. the 1st ACM Symposium on Cloud Computing, Indianapolis, IN, USA, June 10-11, 2010, pp.205-216.

[27] L. C. Canon, E. Jeannot, and J. Weissman, "A scheduling and certification algorithm for defeating collusion in desktop grids," In International Conference on Distributed Computing Systems, pp. 343–352, 2011.

[28] Jyothsna V., Rama Prasad V.V. FCAAIS: Anomaly based network intrusion detection through feature correlation analysis and association impact scale ICT Express, 2016, pp. 103-116.

[29] Mapper API for Google AppEngine. http://googleappengine.blogspot.com /2010/07/introducing-mapper-api.html (site visited January 2016).

[30] Y. Chen, V. Paxson, and R. Katz. 2010. What's New About Cloud Computing Security. Technical Report UCB/EECS-2010-5, Berkeley,2014, pp 20-29.

[31] A. Matsunaga, M. Tsugawa, and J. Fortes. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics. Microsoft eScience Workshop, 2008, pp 222-229.