Agent-Based Autonomous Result Verification Mechanism in Desktop Grid Systems

HongSoo Kim[1], JoonMin Gil[2], ChongSun Hwang[1], HeonChang Yu[3], and SoonYoung Joung[3,⋆]

[1] Dept. of Computer Science & Engineering, Korea University, 1, 5-Ga, Anam-Dong, SungBuk-Gu, Seoul 136-701, Republic of Korea
{hera,hwang}@disys.korea.ac.kr

[2] Dept. of Computer Science Education, Catholic University of Daegu, 330 Geumnak, Hayang-eup, Gyeongsan-si, Gyeongbuk 712-702, Republic of Korea
jmgil@cu.ac.kr

[3] Dept. of Computer Science Education, Korea University, 1, 5-Ga, Anam-Dong, SungBuk-Gu, Seoul 136-701, Republic of Korea
{yuhc,jsy}@comedu.korea.ac.kr

Abstract. In this paper we discuss the design of result verification in desktop grid systems. In this design, correctness and performance are considered as important issues. To guarantee the correctness of work results, sabotage-tolerant mechanisms have been mainly used, such as voting-based schemes and trust-based schemes. However, these mechanisms result in low scalability and high computation delay because they can not cope effectively with dynamic environments. In this paper, we propose a Sabotage-Tolerant Scheduling for Result Verification (STSRV), which is based on mobile agent technology. In STSRV, mobile agents are used to check periodically the credibility and availability of each volunteer. Using credibility and availability information, our desktop grid system can provide correctness of work results without a huge increase in the computation delay caused by result verification. Additionally, simulation results show that STSRV increases turnaround time for works from the viewpoint of credibility and availability, and thus enhances the overall performance of our desktop grid systems.

1 Introduction

Desktop grid computing is a computing paradigm for carrying out high throughput scientific applications by utilizing the idle time of desktop computers (or PCs) connected over the Internet [6]. One of the main characteristics of this kind of computing is that computing nodes, referred to volunteers or workers, are free to leave or join the network, according to their own schedule. In addition, each node is administered autonomously.

Accordingly, Desktop Grid Systems (DGSs) are based on uncontrolled and unspecified computing nodes and thus cannot help being exposed to sabotage

⋆ Corresponding author.

caused by the erroneous results of malicious volunteers. If a malicious volunteer submits bad results to a server, all results can be affected. For example, it has been reported that SETI@Home suffered from malicious behavior by some volunteers who faked the number of work objects completed; some volunteers actually faked their results, by using a code different from the original one [2][11]. Consequently, DGSs should be equipped with sabotage-tolerance mechanisms so as to prevent them from intentional attacks by malicious volunteers.

In existing work, result verification for the results of computation is mainly realized by voting based schemes or trust based schemes. In voting based schemes, the same work is distributed to several nodes (generally, more than three nodes) by replication and the results are returned for comparison with each other. If the same results are received from the majority of the nodes, the result is accepted as final. This scheme is simple and straightforward, but is insufficient because it wastes the resources of computing nodes. On the other hand, trust based schemes [3][10][4][9][8] do not replicate work objects for result verification, and so have a lower redundancy than voting schemes. Instead, a spotter work object (or a sample) is periodically distributed to volunteers, on the assumption that a correct result of the spotter work object is already known. In this way, the server can obtain the trust of each volunteer. This trust is used as a key factor in the scheduling phase. However, these schemes are based on First-Come First-Serve (FCFS) scheduling. From the viewpoint of result verification, FCFS scheduling will cause high computation delay. The reason for this is because it can not cope effectively with dynamic environments such as nodes leaving or joining the network, due to computation interference and crash failures.

To overcome these problems in DGSs, we propose a Sabotage-Tolerant Scheduling for Result Verification (STSRV) with mobile agents based on an autonomous scheduling mechanism. First, we devise Autonomous Sampling with Mobile Agents (ASMA) to evaluate the credibility and availability of each volunteer. In ASMA, a lightweight work object, the correct result of which is already known, is distributed to volunteers. The result computed by each volunteer is compared with the correct result of the object. At this point, the volunteer's availability is also checked. Then, values for credibility and availability are used to classify volunteers and organize the computation group. Secondly we propose Best Credible and Available Volunteer First-Serve (BCAVFS) scheduling, which guarantees correctness of work results and reduces computation delays in result verification. Additionally, simulation results show that STSRV increases turnaround time for work from the viewpoint of credibility and availability, and thus enhances the overall performance under result verification in desktop grid systems.

The rest of this paper is organized as follows. In Section 2, we present the system model used in our DGS. In Section 3, we define protocols and functions for the multiregion mobile agent system. Section 4 describes our result verification mechanism based on the credibility and availability of volunteers. In Section 5, we present implementation and performance evaluation. Finally, our conclusions are given in Section 6.

2 Desktop Grid System Model

Figure 1 shows the overall architecture of our DGS model. As shown in this figure, our DGS model consists of clients, a computation management server (CMS), a storage server, coordinators, and volunteers. A client submits its own application to the CMS. A coordinator takes charge of task scheduling, computation group (CG) management, and agent management. A volunteer is a resource provider that contributes its own resources in order to process large scale applications during idle time periods. Next, our DGS performs several phases as follows:

1. ***Registration Phase***: Volunteers register their information (e.g. CPU, memory, OS type, etc) to a CMS. On receiving the information, the CMS sends it to a coordinator.
2. ***Job Submission Phase***: A client submits a large scale application to a CMS.
3. ***Task Allocation Phase***: A CMS allocates a set of tasks to coordinators after a job is split into a lot of tasks.
4. ***Load Balancing Phase***: A coordinator inspects tasks in the task pool either periodically or on demand and performs load balancing through the migration of some tasks to other coordinators.
5. ***Scheduling Phase***: A coordinator conducts scheduling, by which a task in the task pool is distributed to an available resource.

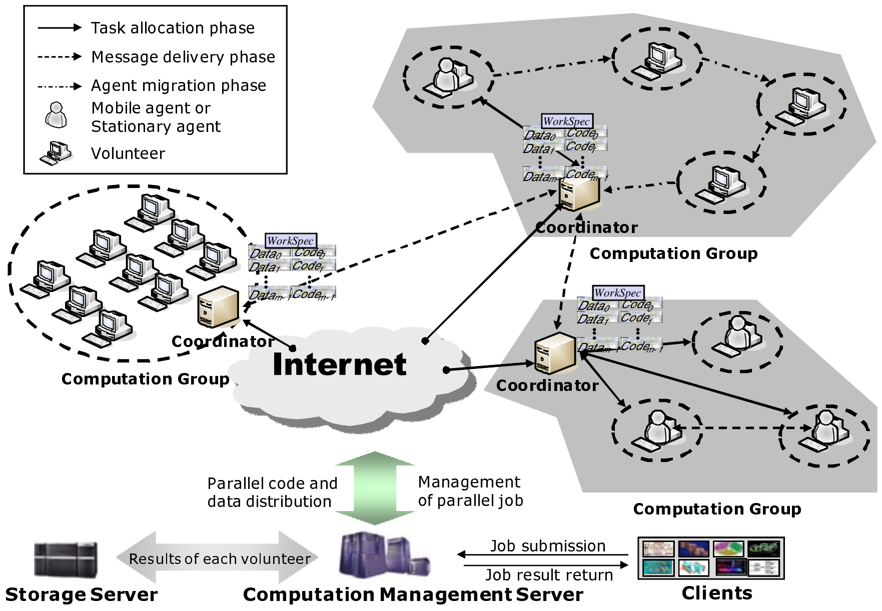


Fig. 1. Desktop grid system environment

6. ***Result Collection Phase*:** A coordinator collects results from volunteers and performs result verification.
7. ***Job Completion Phase*:** A coordinator returns a set of correct results to a CMS.

In this paper, we assume a work-pool based distributed master-worker. This model is suitable for DGSs because of its scalability and reliability characteristics. Under this model, our DGS can use a computation group manager, which operates together with the coordinator.

As shown in Fig. 1, an application is divided into a sequence of batches (or work objects), each of which consists of mutually independent work objects as a set $W = \{w_1, w_2, ..., w_n\}$, where $\boldsymbol{n}$ represents the total number of work objects. We also assume the Single-Program Multiple-Data (SPMD) model, which executes the same code for different data.

In this paper, mobile agent technology is exploited to make the scheduling mechanism adaptive to the dynamic DGS. There are some advantages of making use of mobile agents in DGS. 1) A mobile agent can decrease the overhead of server by performing scheduling, fault tolerance, and replication algorithms in a decentralized fashion. 2) A mobile agent can adapt to dynamical DGS.

3 A Multiregion Mobile Agent System

ODDUGI is a mobile agent system [12] organized into many regions (or computation groups in DGSs). This system consists of five components: the mobile agent, volunteer (or node), computation group (or region), group coordinator (GC), and application server (AS). In this environment, mobile agents execute work (actual samples of a work object) on a sequence of nodes while migrating to nodes in other regions, and stationary agents perform applications (work objects) on nodes.

The ODDUGI mobile agent system has the following design goals. First, the message delivery protocol guarantees that messages should be delivered to mobile agents in a timely manner. That is, the message delivery protocol should be able to deliver a message asynchronously and immediately. Second, scalability is one of the most important design goals when developing distributed grid systems; it enables a system to be scalable without performance degradation. The message delivery protocol should be scalable, even though the number of mobile agents increases. In addition, it should be geographically scalable even though nodes or mobile agents may be dispersed over great distances. Third, mobile agents can possibly fail to deliver messages, due to problems with mobility. Tracking mobile agents is a challenge. In addition, a message cannot be delivered to a mobile agent under going migration. Therefore, a message delivery protocol must solve these problems for reliable message delivery.

3.1 Reliable Message Passing Protocol

Our messaging protocol exploits a blackboard, which is a shared information space for message exchange among agents, and relates migration with message delivery. As a result, it guarantees asynchronous and reliable message delivery to mobile agents.

In our protocol, each GC is responsible for message delivery for all the agents in its own computation group. Each GC has a blackboard where it stores messages from sender agents. Then, when receiver agents update new messages such as current execution state and locations to the GC (after they join in the system), the GC checks the blackboard for messages for the receiver agent. If messages exist, the GC sends them to the receiver agent. Our message delivery protocol is asynchronous but guaranteed and tightly coupled with agent execution in a dynamic environment.

3.2 Autonomous Sampling Using Mobile Agents

Our system periodically performs Autonomous Sampling using Mobile Agents (ASMA) to estimate the credibility and availability of each volunteer. The ASMA scheme is proposed to increase scalability and to guarantee correctness with lower communication cost. Previous work [3][10] has relatively high communication cost because the already known result is compared with the returned result after samples (or spotters) are sent to each volunteer by a central management server. ASMA will cut down the communication cost because the agent migrates to volunteers after being created in a coordinator. The mobile agent then counts the number of samples performed by the volunteer after comparison with sampling result. Subsequently, the agent performs further sampling of the computation group by an itinerary migration.

The ASMA migrates with the itinerary before assigning a work object, and collects the result after performing the sample at each volunteer. To protect against agent failure or system failure in every execution, the sampling result has to be preserved in a local blackboard. When the agent finishes sampling all volunteers, it sends a set of results to the coordinator. On receiving a set of results, the coordinator updates C_i, a credibility value of volunteer i. This procedure is repeated periodically, prior to performing new work objects.

4 Result Verification

This section describes our Sabotage-Tolerant Scheduling for Result Verification (STSRV) mechanism.

4.1 Overview

Basically, our approach applies a replication policy to volunteer with low credibility and low availability, in order to guarantee correctness simultaneously with performance. For that reason, the scheduler compares C_i, the credibility value of a volunteer $\boldsymbol{i}$, with a credibility threshold θ. If $C_i > \theta$, then the work object should be allocated to volunteer $\boldsymbol{i}$. Otherwise, the scheduler selects the volunteer of the next C_i value, and determines replication by comparing C_i with θ. The

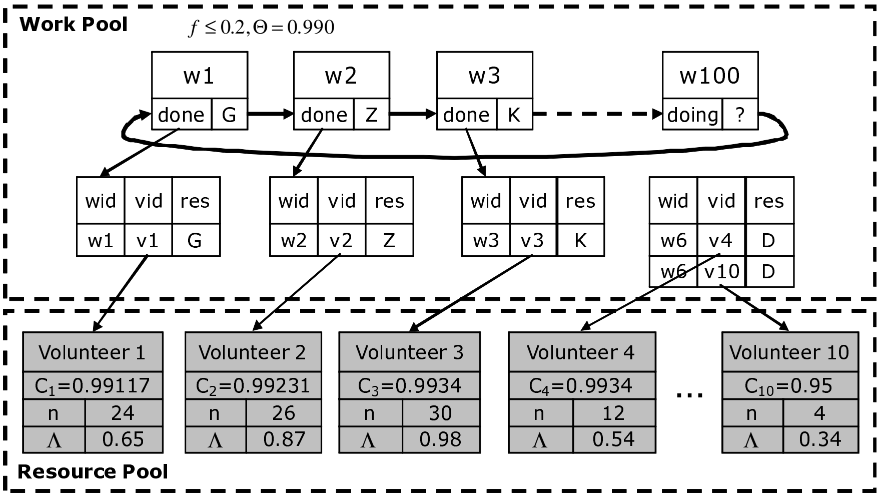

Work Pool
$f \leq 0.2, \Theta = 0.990$
w1 done G
w2 done Z
w3 done K
w100 doing ?
wid vid res
w1 v1 G
w2 v2 Z
w3 v3 K
w6 v4 D
w6 v10 D
Volunteer 1 C1=0.99117 n 24 Λ 0.65
Volunteer 2 C2=0.99231 n 26 Λ 0.87
Volunteer 3 C3=0.9934 n 30 Λ 0.98
Volunteer 4 C4=0.9934 n 12 Λ 0.54
Volunteer 10 C10=0.95 n 4 Λ 0.34
Resource Pool

Fig. 2. Work pool model based on credibility and availability to result verification in each coordinator

credibility threshold θ has a different value for each application and is calculated as $\theta = 1 - \varepsilon$, where ε is an acceptable error rate.

Figure 2 shows a work pool and a resource pool for a single coordinator. The work pool has a set of work objects, denoted by w_{id}, a volunteer denoted by v_{id} and ***res***, a result value. The resource pool includes credibility values of volunteers C_i, a sampling number ***n***, and volunteer availability Λ_i. For example, if the scheduler assigns a work object w_3 to a volunteer v_3 where C_3 is 0.99349, the scheduler assigns the work object to the volunteer without any replication because of high credibility threshold.

In DGSs, the executing applications each have a different error rate ***f***. The executed result by a volunteer could be accepted if the results verify the reliability of a voting scheme. First of all, when a scheduler assigns a work object, it executes concurrent assignments according to the number of replications (refer to subsection 4.3). Moreover, the scheduler determines redundancy number, according to the credibility value, error rate and the availability value.

4.2 Definition and Classification

In the desktop grid computing environment, system performance is influenced by the dynamic nature of volunteers [13]. Accordingly, the availability of a volunteer has to be considered for performance improvement as well as for reliability of computation execution. Moreover, there is a need to utilize the credibility of volunteers to guarantee correctness for the results executed by volunteers. Therefore, a volunteer classification scheme based on credibility and availability is also needed. As shown in Fig. 3, we classify volunteers into four types according to credibility and availability. The following shows the definition of credibility and availability.

***Volunteer Credibility*(*C*).** The credibility is a factor determining correctness for the result of the computation executed by a volunteer.

$$C_i = \begin{cases} 1 - \frac{f}{n}, & \text{if } \boldsymbol{n} > 0 \\ 1 - f, & \text{if } \boldsymbol{n} = 0 \end{cases} \tag{1}$$

In Equation (1), C_i represents the credibility of a volunteer v_i, $\boldsymbol{n}$ is the number of correctly results returned by the ASMA scheme, and failure rate $\boldsymbol{f}$ is the probability that a volunteer chosen at random is bad. If $\boldsymbol{n}$ is 0, then it would be calculated as 1-$\boldsymbol{f}$.

***Volunteer Availability*(Λ).** The availability is the probability that a volunteer will participate in a computation failure.

$$\Lambda_i = \frac{MTTCF}{MTTCF + MTTCR} \tag{2}$$

In Equation (2), ***MTTCF*** means the mean time to computation failure, which indicates how often a volunteer leaves during computation. ***MTTCR*** represents the mean time to computational repair, which means the average time for the volunteer to join.

In this paper, volunteers are classified into four types according to availability and credibility as Fig. 3. Affirmative Volunteers (AVs) represent volunteers with high credibility and high availability. These volunteers can be used as a substitution node for other volunteers with low credibility or low availability. Passive Volunteers (PVs) represent volunteers that have high credibility and low availability. These volunteers can guarantee high availability through replication. Active Volunteers (A'Vs) represents volunteers that can achieve result verification through random sampling, because of low credibility and high availability. Finally, Negative Volunteers (NVs) are not reliable and secure due to low credibility and low availability, and thus their work is transferred to volunteers of the other three types.

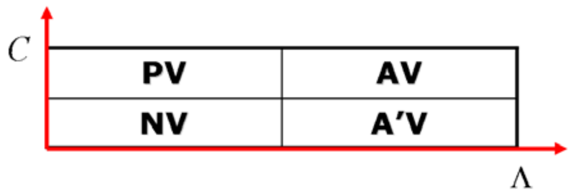


Fig. 3. Classification of volunteers according to volunteer availability (Λ) and volunteer credibility (C_i)

4.3 Computation Replication

In a desktop grid environment, task completion rate depends on the availability of individual volunteers. Replication is one of the methods to achieve a high completion rate. Its aim is to cope with the failure of volunteers together with reliable computation. Replication in our DGS determines the number of replicas according to volunteer availability and credibility. Each volunteer records its own task execution time as an execution history into a coordinator's repository. Based on the execution history, the average availability of each volunteer ($\overline{\Lambda_i}$) during some period from past time to current is calculated by

$$\overline{\Lambda_i} = \frac{\sum \Lambda_k^{'}}{K} \tag{3}$$

where, $\Lambda_k^{'}$ represents the kth task's execution time and K is the number of tasks completed by volunteer i.

Using Equation (3), the number of replicas is determined by

$$n = \frac{\overline{\Lambda_i}}{\Psi_i \times C_i} \tag{4}$$

where, Ψ_i represents the execution time of a task without a computation failure in volunteer i and C_i represents the credibility of volunteer i.

Using Equation (4), our DGS can determine the number of replicas distributed to volunteers. Once computation in a volunteer is completed, computation results are returned to the coordinator.

5 Sabotage-Tolerant Scheduling

When as many resources are selected as the number of replicas, our DGS performs BCAVFS scheduling, based on volunteers' credibility and availability. In general, our scheduler performs several steps. The detailed sequence is as follows:

1. The system is initiated when it sends the "start" event to a scheduler.
2. The scheduler obtains "undone" work objects from the task pool.
3. After determining the number of replicas by Equation (4), the scheduler allocates the incomplete work objects to redundant volunteers with the best credibility and availability, $MAX(C_i * \Lambda_i)$, in the resource pool.
4. On completing the allocated work object, a volunteer returns its result to the scheduler. If computation for whole replicas is finished, the scheduler performs result verification. Then, the scheduler updates the resource pool and task pool.
5. The scheduler notifies the coordinator of the completion of computation, and the coordinator requests new tasks from a CMS.
6. Steps 2-5 are repeated until there are no more work objects.

6 Implementation and Performance Evaluation

Our BCAVFS scheduling was experimentally implemented in the Korea@Home [7] DGS operating with the ODDUGI mobile agent system [12]. The Korea@Home DGS aims to harness the massive computing power by utilizing the huge number of desktop computers connected over the Internet. The ODDUGI developed with J2SE 1.4 is a mobile agent system supporting reliable, secure and fault tolerant execution of mobile agents. Fig. 4 shows the overall system organization for the "Korea@Home" DGS operating with the "ODDUGI" mobile agent system. Our system consists of application, context, and runtime layers, as shown in Fig. 4. The runtime layer initiates the context and various managers such as resource manager, security manager, location manager, and message manager. The context layer provides a logical execution environment for the desktop grid and mobile agents. It provides core functionalities such as scheduling, monitoring, creation, execution, clone, migration, retraction, activation, deactivation, and termination. The enhanced mechanisms such as fault tolerance, security, location management, and message delivery are implemented in this layer. Finally, the application layer provides APIs for grid applications and GUIs for system managers.

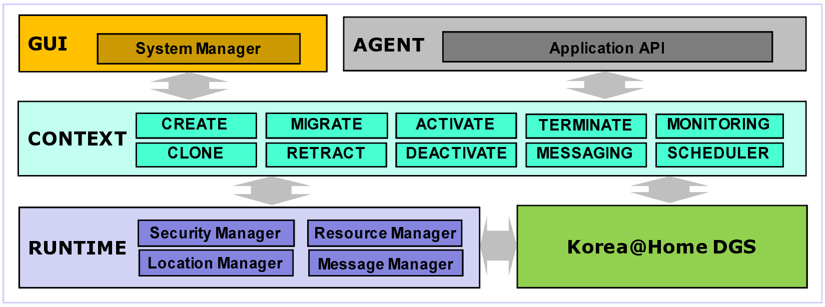
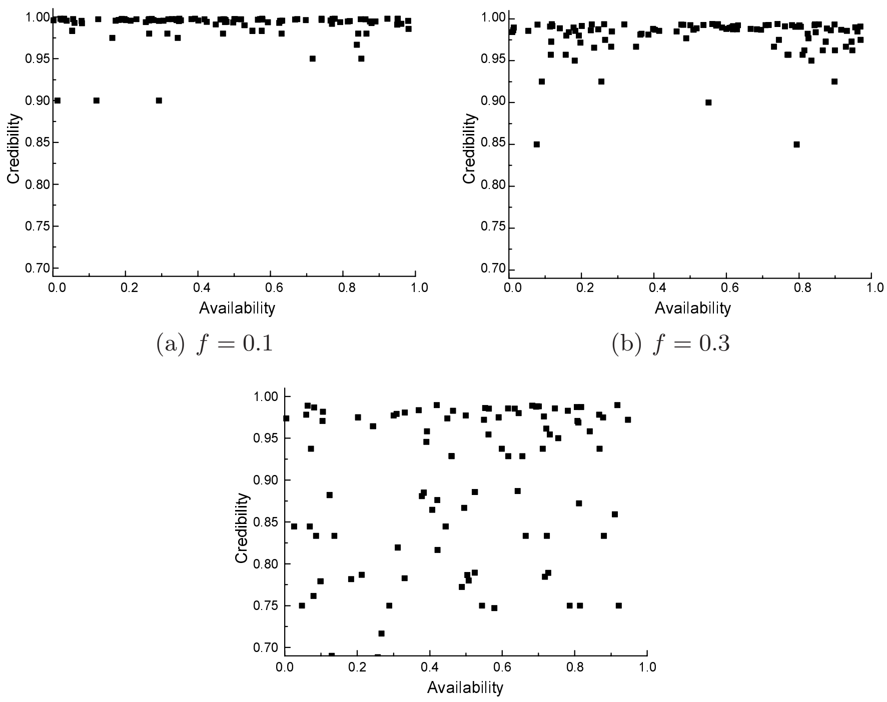


Fig. 4. System organization for "Korea@Home" DGS operating with "ODDUGI" mobile agent system

Our simulation was conducted with actual volunteers in the Korea@Home DGS.The application used in our simulation was a new drug candidate discovery based on virtual screening technology in the field of bioinformatics. A task in the application consumes approximately 16 minutes of execution time on a dedicated Pentium 1.4 GHz. A total of the 728 volunteers participated in our simulation.

Fig. 5 shows the distribution of volunteers according to credibility and availability when the error rate f is 0.1, 0.3, and 0.5, respectively. In this figure, x and y axes represent availability values ranging from 0.0 to 1.0 and credibility values ranging from 0.7 and 1.0, respectively. From this figure, we can observe that as the error rate becomes higher, volunteers are more spread over two dimensional spaces of credibility and availability. For example, as we can see in Fig. 5(c), most volunteers are dissimilar from the viewpoint of credibility and availability.



(a) $f = 0.1$ (b) $f = 0.3$

(c) $f = 0.5$

Fig. 5. Distribution of volunteers according to credibility and availability

Using the distribution results of Fig. 5, we classified the four volunteer types (***AV***, ***PV***, ***A'V***, and ***NV***) into three cases. Table 1 shows the simulation environment with different credibility and availability values. In this table, Case 1 means that credibility and availability are relatively lower than other cases. On the other hand, Case 3 means that credibility and availability are relatively higher than other cases.

As stated in [13], turnaround time is an important performance measurement in DGS. Fig. 6 shows turnaround time for three cases when f is 0.1, 0.3, and 0.5. From this figure, we can observe that turnaround time is greatly affected by error rate; as the error rate increases, the turnaround time for all the three cases grows. This is because a high error rate results in a reduction in the number of completion tasks, leading to low credibility. On the other hand, a low error rate results in high credibility. Thus, most tasks can be returned quickly with few computation failures. On analyzing turnaround time for each case, we can see that Case 3 has lower turnaround time than the other two cases. It should be noted that Case 3 constructs a computation group belonging to volunteers with high credibility rather than other cases. On the whole, low error rate and high credibility result in fast turnaround time in our DGS.

Table 1. Simulation environment

Cases	Parameters	NV	PV	$A'V$	AV
Case 1	$C(\%)$	0.0-0.9	0.9-1.0	0.0-0.9	0.9-1.0
	$\Lambda(\%)$	0.0-0.5	0.0-0.5	0.5-1.0	0.5-1.0
Case 2	$C(\%)$	0.0-0.95	0.95-1.0	0.0-0.95	0.95-1.0
	$\Lambda(\%)$	0.0-0.7	0.7-1.0	0.0-0.7	0.7-1.0
Case 3	$C(\%)$	0.0-0.98	0.98-1.0	0.0-0.98	0.98-1.0
	$\Lambda(\%)$	0.0-0.9	0.9-1.0	0.0-0.9	0.9-1.0

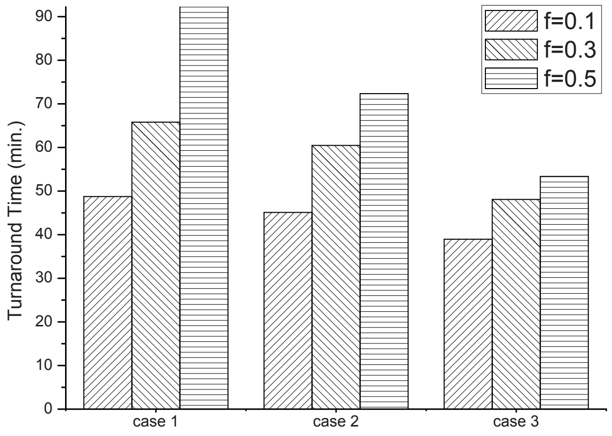


Fig. 6. Turnaround time

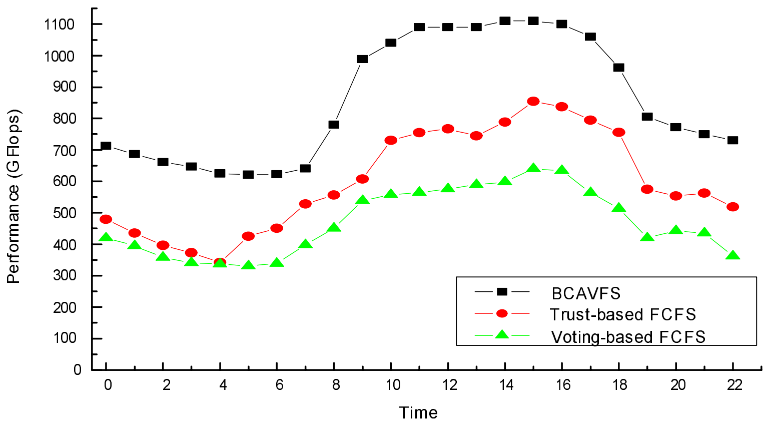


Fig. 7. Performance comparison of BCAVFS with trust-based and voting-based FCFS for one day

Next, we measured the performance of our scheduling mechanism. For performance evaluation, each agent sends its own work information to a coordinator whenever an event occurs. The coordinator measures the performance of our DGS by Linpack benchmark [14] in a unit of one hour. Fig. 7 shows the performance comparison of our BCAVFS with other two FCFS scheduling methods, voting-based FCFS and trust-based FCFS for one day. As we can see in this figure, our BCAVFS is about 743 GFlops on average, but on the other hand trust-based FCFS is about 596 GFlops, and voting-based FCFS is 474 GFlops on average, respectively. From this figure, we can see that our BCAVFS is superior to the other FCFS schedulers to result verification.

7 Conclusion

In this paper, we proposed the STSRV mechanism to guarantee result correctness and improve performance in DGS. In our performance evaluation, we used credibility and availability of actual volunteers from Korea@Home DGS. The performance results showed that our scheduling is superior to voting-based FCFS scheduling and trust-based FCFS scheduling.

In the near future, various metrics for result verification will be applied in the proposed scheduling mechanism. We also plan to develop a scheduling mechanism based on probability model in dynamic desktop grid computing environment.

Acknowledgment

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2007-324-D00223).

References

1. Neary, M.O., Cappello, P.: Advanced Eager Scheduling for Java Based Adaptively Parallel Computing. Concurrency and Computation: Practice and Experience 17(7-8), 797–819 (2005)
2. Molnar, D.: The SETI@home Problem, `http://turing.acm.org/crossroads/columns/onpatrol/september2000.html`
3. Sarmenta, L.: Sabotage-Tolerance Mechanism for Volunteer Computing Systems. Future Generation Computer Systems 18(4), 561–572 (2002)
4. Germain-Renaud, C., Playez, N.: Result Checking in Global Computing Systems. In: Proc. of the 17th Annual Int. Conf. on Supercomputing, June 2003, pp. 226–233 (2003)
5. Neary, M.O., Phipps, A., Richman, S.: Javelin 2.0: Java-Based Parallel Computing on the Internet. In: Bode, A., Ludwig, T., Karl, W.C., Wismüller, R. (eds.) Euro-Par 2000. LNCS, vol. 1900, pp. 1231–1238. Springer, Heidelberg (2000)
6. Germain, C., Fedak, G., Neri, V., Cappello, F.: Global Computing Systems. In: Margenov, S., Waśniewski, J., Yalamov, P. (eds.) LSSC 2001. LNCS, vol. 2179, pp. 218–227. Springer, Heidelberg (2001)

7. Korea@Home homepage, http://www.koreaathome.org
8. Azzedin, F., Maheswaran, M.: A Trust Brokering System and Its Application to Resource Management in Public-Resource Grids. In: Proc. of the 18th Int. Parallel and Distributed Processing Symposium, April 2004, pp. 22–31 (2004)
9. Du, W., Jia, J., Mangal, M., Murugesan, M.: Uncheatable Grid Computing. In: Proc. of the 24th Int. Conf. on Distributed Computing Systems, pp. 4–11 (2004)
10. Zhao, S., Lo, V., Dickey, C.G.: Result Verification and Trust-Based Scheduling in Peer-to-Peer Grids. In: Proc. of the 5th IEEE Int. Conf. on Peer-to-Peer Computing, September 2005, pp. 31–38 (2005)
11. SETI@Home homepage, http://setiathome.ssl.berkeley.edu
12. Choi, S., Baik, M., Kim, H., Byun, E., Hwang, C.: Reliable Asynchronous Message Delivery for Mobile Agent. IEEE Internet Computing 10(6), 16–25 (2006)
13. Kondo, D., Taufer, M., Brooks, C.L., Casanova, H., Chien, A.: Characterizing and Evaluating Desktop Grids: An Empirical Study. In: Proc. of the 18th Int. Parallel and Distributed Processing Symposium, April 2004, pp. 26–35 (2004)
14. Dongarra, J.: Performance of various computers using standard linear equations software. ACM SIGARCH Computer Architecture News 20, 22–44 (1992)