Future Generation Computer Systems 26 (2010) 776–786

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Group-based adaptive result certification mechanism in Desktop Grids

SungJin Choi *, Rajkumar Buyya

Department of Computer Science and Software Engineering, University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 1 August 2008
Received in revised form
2 March 2009
Accepted 13 May 2009
Available online 11 June 2009

Keywords:
Desktop grid
Result certification
Reliability
Volunteer computing

ABSTRACT

In Desktop Grids, volunteers (i.e, resource providers) have heterogeneous properties and dynamically join and leave during execution. Moreover, some volunteers may behave erratically or maliciously. Thus, it is important to detect and tolerate erroneous results (i.e., result certification) in order to guarantee reliable execution, considering volatility and heterogeneity in a scheduling procedure. However, existing result certification mechanisms do not adapt to such a dynamic environment. As a result, they undergo high overhead and performance degradation.

To solve the problems, we propose a new Group-based Adaptive Result Certification Mechanism (GARCM). GARCM applies different result certification and scheduling algorithms to volunteer groups that are constructed according to their properties such as volunteering service time, availability and credibility.

1. Introduction

A Grid system is a platform that provides access to various computing resources owned by institutions by making virtual organizations [15,16]. In contrast, a Desktop Grid system is a platform that achieves high throughput computing by harvesting a number of idle desktop computers owned by individuals at the edge of the Internet using peer to peer computing technologies [17–25]. The Desktop Grid systems[1] usually support embarrassingly parallel applications that consist of a lot of instances of the same computation with each data. The applications are usually involved with scientific problems requiring large amounts of processing capacity over long periods of time. Recently, there has been a rapidly growing interest in Desktop Grid systems because of the success of the most popular examples, i.e., GIMPS [26], distributed.net [27] and SETI@Home [28].

A Desktop Grid computing environment mainly consists of client, volunteer and server, as shown in Fig. 1. A *client* is a parallel job submitter who requests results. A *volunteer* is a resource provider that donates its idle computing resources. A *server* is a central manager that controls submitted jobs and volunteers. It can have a file server to maintain tasks and results files. A client first submits a parallel job to a server. The job is then divided into sub-jobs that have their own specific input data. The sub-job is called a *task*. The server distributes tasks to volunteers using scheduling mechanisms. Each volunteer executes its task when idle. When each volunteer subsequently finishes its task, it returns the result of the task to the server. Finally, the server returns the final result of the job back to the client.

A Desktop Grid computing is complicated by heterogeneous capabilities, failures, volatility (i.e., intermittent presence), and lack of trust [24,25,29] because it is based on desktop computers (i.e., volunteers) at the edge of the Internet. In particular, volunteers at the edge of the Internet are exposed to link and crash failures. In addition, they can dynamically join and leave in the middle of an execution without impediment. Thus, *public execution* (i.e., the execution of a task as a volunteer) may be halted arbitrarily. Moreover, volunteers are not dedicated exclusively to Desktop Grid computing, so public executions become temporarily suspended by a *private execution* (i.e., the execution of a private job as a personal user). In this paper, we regard such unstable situations as *volunteer autonomy failures* because they lead to delay and blocking of task execution and even partial or entire loss of the execution. The volunteer autonomy failures occur more frequently than in a Grid computing environment because a Desktop Grid computing system is based on dynamic desktop computers. Volunteers have different occurrence rates for *volunteer autonomy failures* according to their execution behavior. In addition, since any node can participate as a volunteer (i.e., resource provider), some malicious volunteers tamper with the computation and return corrupted results. Therefore, Desktop Grid computing systems

* Corresponding address: GRIDS Lab., Department of Computer Science & Software Engineering, The University of Melbourne, 5.21(Room), 111 Barry street, Carlton, VIC 3053, Australia. Tel.: +61 3 8344 1347; fax: +61 3 9348 1184.
E-mail addresses: lotieye@gmail.com (S. Choi), raj@csse.unimelb.edu.au (R. Buyya).
URL: http://www.cs.mu.oz.au/~lotieye/ (S. Choi).

[1] Desktop Grids are also called volunteer computing [20], global computing [21], public resource computing [19], or Peer-to-Peer Grid computing [25].

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.05.025

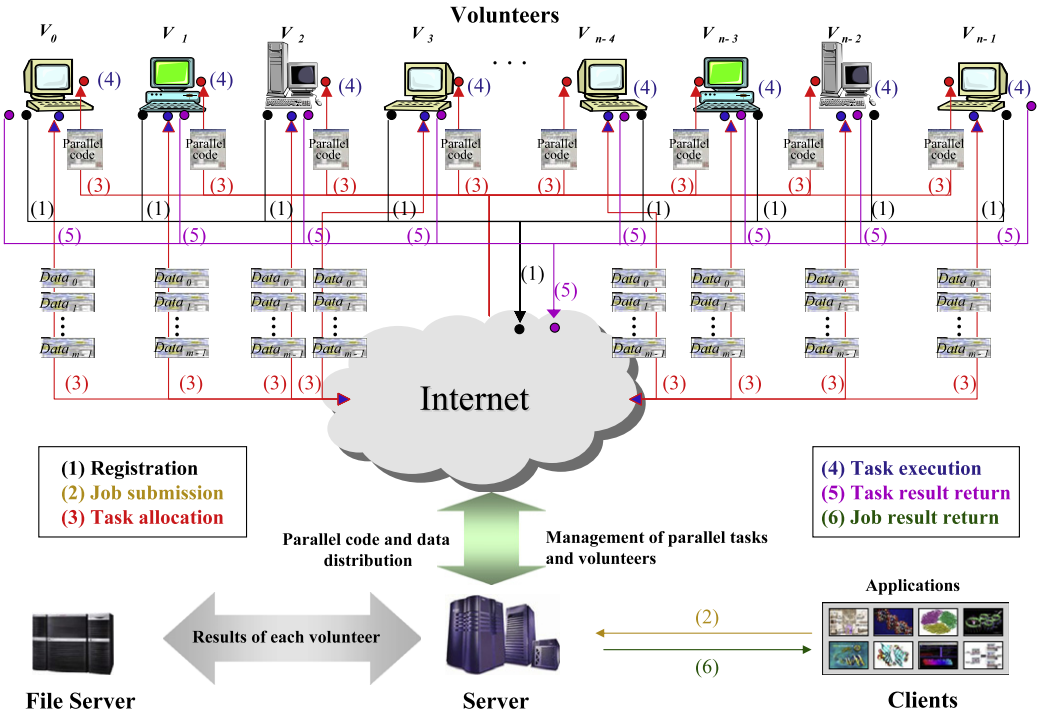


Fig. 1. Desktop Grid computing environment.

must not only adapt to such a dynamic environment, but also detect and tolerate the erroneous result (i.e., result certification) in order to guarantee reliable execution in such a distrustful environment.

To this end, existing Desktop Grid systems exploited result certification mechanisms such as voting and spot-checking [30,1–14,31]. However, (i) existing result certification mechanisms do not adapt to the distinct features resulting from the heterogeneous properties and volatility; (ii) There is no dynamic scheduling for result certification although the result certification is tightly related with scheduling in that both the special task for spot-checking and the redundant tasks for voting are allocated to volunteers in a scheduling procedure; (iii) Existing Desktop Grid systems simply used the eager scheduling mechanism [32,20,33,21,2], although the result certification and scheduling mechanisms are required to classify volunteers into groups that have similar properties, and then dynamically apply various scheduling mechanisms to each group. As a result, they suffer from high overhead and performance degradation because they do not adapt to a dynamic Desktop Grid computing environment.

To solve these problems, we propose a new Group-based Adaptive Result Certification Mechanism (GARCM) that adapts to a dynamic Desktop Grid computing environment. GARCM also provides dynamic scheduling algorithms for result certification. It dynamically performs result certification and scheduling algorithms to each group according to its properties such as volunteering service time, availability, and credibility. To this end, it constructs *volunteer groups* that are classified on the basis of the volunteer's properties, and applies appropriate scheduling and result certification algorithms to each volunteer group. Consequently, GARCM can reduce the overhead and latency and therefore complete more tasks while guaranteeing reliable results.

The rest of the paper is structured as follows. Section 2 presents a new taxonomy of result certification and describes limitations of existing scheduling for result certification in Desktop Grid. Section 3 describes GARCM in details. Section 4 presents the analysis of result certification mechanisms and experimental results. Section 5 concludes the paper.

2. Background and motivation

2.1. A taxonomy of result certification

Result certification aims to detect and tolerate erroneous results in order to guarantee a trusted execution. Some works have focused on result certification in a Desktop Grid computing environment [25,29,30,1–13]. A new taxonomy of result certification is presented in terms of a judgment method, a comparison object, a resource selection method and response to worker's behavior, as shown in Fig. 2.

Result certification is categorized into two approaches according to the judgment method: voting and spot-checking (or sampling). In the *voting approach*, the same tasks are distributed to different volunteers (that is, voting group) as many as the number of redundancies or until the predefined threshold is reached. Redundancy makes it possible to identify the correct result against an erroneous one if there are sufficiently more good volunteers than bad ones. When the results are returned from volunteers (or workers), they are compared to verify the correctness. In majority voting, if the majority of volunteers return the same result, it is regarded as the correct one. In the threshold-based weighted voting, if the results reach the predefined threshold (for example, error rate, credibility and reliability), they are considered trustworthy. Weighted voting uses the reputation of volunteers (for example, credibility, reliability and trust) to compare the results. If volunteers have a higher reputation, they are considered to be more trustworthy. For example, the votes of highly-reputed volunteers carry more weight than others. In the *spot-checking approach*, the special tasks with different inputs, whose result is already known (for example, one-way hash function and quiz), are distributed to volunteers. When the results are returned from volunteers, they are compared with the already-known result produced by a reliable resource (that is, the oracle). If volunteer's response is different (in the simple spot-checking) or the error rate is not within the error-tolerant range or does not reach the predefined threshold (in the threshold-based spot-checking or the threshold-based weighted spot-checking), the volunteer is regarded as

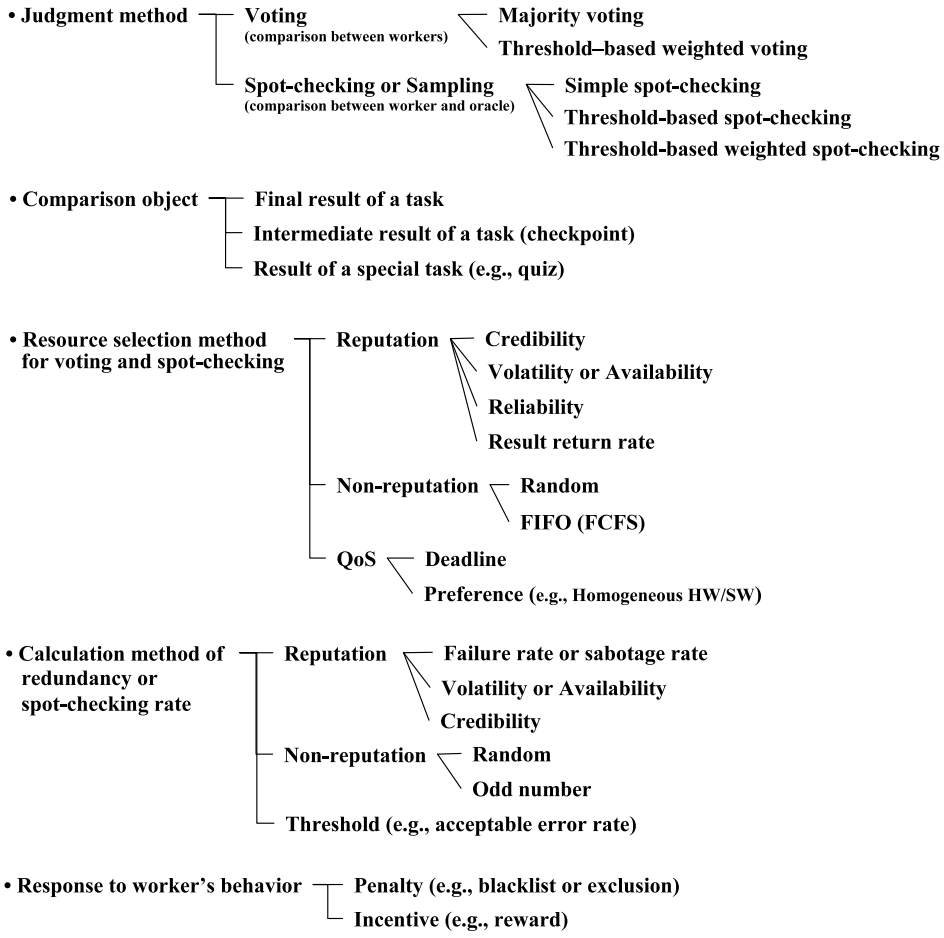


Fig. 2. A taxonomy of result certification.

malicious and the results returned are discarded. A comparison procedure of the voting approach, which verifies whether the result is correct or not, is performed between workers. On the other hand, the spot-checking approach compares worker and the oracle. Thus, it needs a reliable resource (that is, the oracle) to generate a reliable canonical result. The voting approach is apparently more costly than the spot-checking approach, because it requires a redundancy of at least two per task. Spot-checking and voting can be combined together.

Result certification is classified according to the object compared: the final result of a task, the intermediate result of a task and the result of a special task. The final result of a task is mainly used in the voting approach. The intermediate result of a task (that is, a checkpoint) is mainly used for long-running applications in the voting approach in order to detect errors earlier. Additional procedures are needed for the checkpoint approach, for example, how many or often to do checkpoint or how to synchronize several checkpoints that volunteers take at different times due to volatility. The result of a special task (for example, a quiz or a one-way hash function) is used in the spot-checking approach.

Result certification accompanies resource selection methods to choose volunteers for a voting group or for spot-checking, and is categorized according to the resource selection method: reputation, non-reputation and QoS. The reputation-based resource selection chooses volunteers according to their reputation, based on criteria such as credibility, volatility, reliability and result return rate. For example, highly-reputed volunteers are first selected or badly-reputed volunteers are excluded from a scheduling procedure. Non-reputation-based resource selection chooses volunteers randomly or on a FCFS (First Come First Served) basis. Volunteers can be selected according to QoS such as the deadline of a task or homogeneous hardware and software types and versions.

Result certification can be classified according to the calculation method of redundancy or spot-checking rate. It is very important how many redundancies for voting are needed or how often spot-checking is performed because it is directly related to overhead, performance and correctness. First, the number of redundancies for voting and a spot-checking rate can be calculated depending on volunteer reputation such as failure rate, sabotage rate, volatility and credibility. For example, the higher the reputation, the lower the redundancy and spot-checking rate. Second, they can be decided randomly or as an odd number for majority voting. Finally, voting and spot-checking can be performed until the threshold (for example, acceptable error rate) is satisfied.

Result certification can be categorized according to a positive or negative response to worker behavior: incentive and penalty, respectively. Incentive-based responses aim to encourage resource owners to donate their resources reliably and trustworthily by giving rewards (for example, money, resources, ranking, high priority in a scheduling procedure, etc.) to volunteers for their donation. Penalty-based responses attempt to inflict punishment (for example, blacklist, exclusion from scheduling, decrease in reputation and ranking, etc.) on untrustworthy volunteers.

2.2. Motivation

A few studies have been made on scheduling for result certification in a Desktop Grid computing environment [2,9–11,14,31]. Bayanihan [2] proposed majority voting and spot-checking

based on an eager scheduling algorithm. Especially, a credibility-enhanced eager scheduling algorithm is also proposed. In such algorithms, the more a volunteer passes spot-checking, the higher its credibility. The more volunteers within a voting group agree on a result, the higher its credibility becomes. Volunteers continue to compute the task and perform spot-checking until the credibility threshold is satisfied. When the desired credibility threshold is reached, the result is accepted as final. In these algorithms, the voting group for majority voting is not built before tasks are distributed to volunteers. Instead, it is built on the fly. Whenever a faster volunteer is allocated a task, it is added to the voting group for the task. As a result, the members in the voting group have different credibility. Taufer et al. [9] proposed homogeneous redundancy to tolerate variations in numerical processing due to a variety of hardware and software malfunctions. Homogeneous redundancy dispatches redundant tasks to numerically identical volunteers. Zhao et al. [10] proposed result verification and trust-based scheduling mechanisms, combining a reputation system with the basic verification schemes such as majority voting and quiz. Reputation systems maintain trusted and blacklists not only to select trusted volunteers, but also to exclude malicious volunteers from a scheduling procedure. Sonneck et al. [11] proposed an adaptive reputation-based scheduling that uses worker reliability for efficient task allocation. Worker reliability is calculated based on the number of correct results. Reputation-based scheduling attempts to form redundancy groups (that is, voting groups) that satisfy the likelihood of correctness (LOC), which means group workers will return a majority of correct and timely results. The redundancy group can be organized randomly or in order of reliability. Kim et al. [14] proposed a priority-based list scheduling mechanism for sabotage tolerance with deadline tasks. The scheduling mechanism selects volunteers according to credibility and result return probability, satisfying both the correctness of results and task deadlines.

However, there are some limitations in existing scheduling mechanisms for result certification.

(1) *They do not adapt to dynamic Desktop Grid computing environment*. Existing scheduling and result certification mechanisms do not take into account heterogeneous properties and volunteer autonomy failures in scheduling and result certification procedures (i.e., when deciding the number of redundancy and spot-checking rate, or when selecting new volunteers for failed or slow volunteers). As a result, they require more redundancy to achieve result certification as well as more overhead. They cannot complete result certification because of delay and blocking of task execution.
(2) *There are no dynamic scheduling mechanisms for result certification*. Result certification is tightly related with scheduling in the sense that both the special task for spot-checking and redundant tasks for voting are allocated to volunteers in a scheduling procedure. In the presence of failure, the failed tasks are reallocated to new volunteers in a fault tolerant scheduling procedure. However, existing Desktop Grid computing systems [19,32,20,33,21] simply use an eager scheduling mechanism, which is not appropriate for result certification. For example, consider a voting group built on the fly in Bayanihan [2]. When a volunteer is allocated a task, it is added to the voting group for the task. At this time, eager scheduling simply selects the fastest volunteer as a member of the voting group for the task without considering the credibility of the volunteer. As a result, the credibility of the voting group fluctuates. Although a volunteer with a high credibility makes the credibility of the voting group higher, another volunteer with low credibility can reduce the credibility. Thus, it becomes difficult to agree on the same result. In other words, more volunteers are needed to reach agreement. Consequently, delays and overhead problems arise. In the case of failures, when the failed volunteer is replaced with a new volunteer, the same problems also arise.
(3) *They use only one scheduling mechanism at a time statically*. In Desktop Grid computing environments, volunteers have various properties, e.g., capacity, location, availability, credibility, etc. Thus, various scheduling and result certification mechanisms should be dynamically applied at the same time according to the volunteer's properties. However, existing mechanisms use only one mechanism at a time. The same scheduling, result certification, and fault tolerant algorithms are applied to all volunteers without considering their individual properties. As a result, they experience high overhead and scalability problems.

To overcome these limitations, we propose a new group-based adaptive result certification mechanism on the basis of a volunteer group constructed according to volunteering service time, volatility, availability and credibility. The proposed mechanism applies different scheduling and result certification algorithms to volunteer groups according to their properties.

3. Group-based Adaptive Result Certification Mechanism

Group-based Adaptive Result Certification Mechanism (GARCM) dynamically performs different scheduling and result certification algorithms suitable for volunteer groups that are classified on the basis of the properties of volunteers such as *volunteer availability*, *volunteering service time* and *volunteer credibility*. First, we present the construction of a volunteer group according to volunteer properties. Second, we illustrate the application of scheduling and result certification algorithms to volunteer groups, that is, how to select volunteers for voting and spot-checking and how to calculate the redundancy or spot-checking rate according to volunteer group's properties.

3.1. Constructing volunteer groups

Volunteers are required to first be formed into homogeneous groups in order to apply different scheduling and result certification algorithms suitable for volunteers during the scheduling phase. GARCM classifies volunteers into four volunteer groups on the basis of *volunteer availability* α_v, *volunteering service time* Θ, and *volunteer credibility* C_v, that is, A', B', C', and D' volunteer groups. In the A' volunteer group, all features are high. In the B' volunteer group, both α_v and C_v are high. In the C' volunteer group, only Θ is high. Finally, in the D' volunteer group, all features are low.

3.1.1. Classifying volunteers

When classifying volunteers, their CPU and memory capacities are important factors. The most important factors, however, are volunteering time, volunteer availability and credibility because a Desktop Grid computing system is based on dynamic desktop computers. In a Desktop Grid computing environment, the capacities of desktop computers are different, while the volunteering time, availability, and credibility are very diverse [19,34,24,25]. Therefore, the computation time is more affected by the latter factors. In this paper, we classify volunteers according to volunteer availability and volunteering service time. Volunteering time, volunteer availability and volunteering service time are defined as follows.

Definition 1 (*Volunteering Time*). Volunteering time (Υ) is the period when a volunteer is supposed to donate its resources.

$$\Upsilon = \Upsilon_R + \Upsilon_S.$$

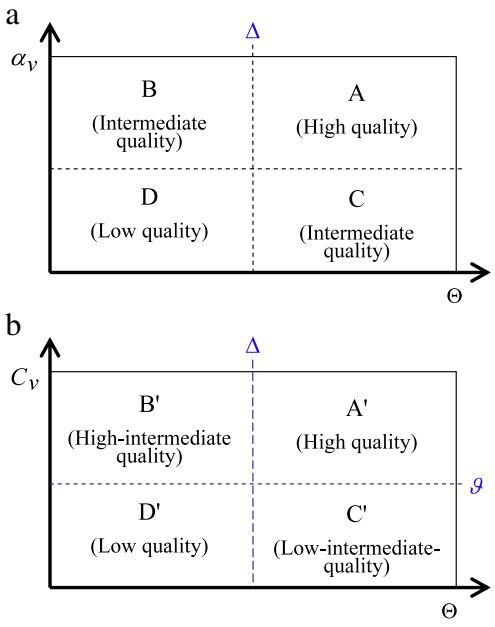


Fig. 3. The classification of volunteers and volunteer groups.

Here, *reserved volunteering time* (Υ_R) is the reserved time when a volunteer provides its computing resources. Volunteers mostly perform public execution during Υ_R, rarely performing private execution. On the other hand, *selfish volunteering time* (Υ_S) is unexpected volunteering time. Thus, volunteers usually perform private execution during Υ_S, while occasionally performing public execution.

Definition 2 (*Volunteer Availability*). Volunteer availability (α_v) is the probability that a volunteer will be correctly operational and be able to deliver volunteer services during the volunteering time Υ

$$\alpha_v = \frac{MTTVAF}{MTTVAF + MTTR}.$$

Here, *MTTVAF* means "mean time to volunteer autonomy failures", and *MTTR* means "mean time to rejoin", MTTVAF means the average time before volunteer autonomy failures happen, and MTTR means the mean duration of volunteer autonomy failures. α_v reflects the degree of volunteer autonomy failure, whereas the traditional availability in distributed systems is mainly related with crash failure.

Volunteering service time is defined as follows.

Definition 3 (*Volunteering Service Time*). Volunteering service time (Θ) is the expected service time when a volunteer participates in public execution during Υ

$$\Theta = \Upsilon \times \alpha_v.$$

In a scheduling procedure, Θ is more appropriate than Υ because Θ represents the time when a volunteer actually executes each task in the presence of volunteer autonomy failures.

Volunteers are categorized into four classes (i.e., *A*, *B*, *C*, *D* classes) according to α_v and Θ, as shown in Fig. 3(a).

3.1.2. Classifying and making volunteer groups

A server selects volunteers as volunteer group members according to volunteer properties such as volunteering service time and volunteer credibility.

Volunteer credibility is defined as follows.

```
// V_A : A class, V_B: B class, V_C : C class, V_D: D class
// VG_A' : A' class, VG_B': B' class, VG_C' : C' class, VG_D': D' class
// To classify the registered volunteers into A, B, C, D classes, respectively
ClassifyVolunteers(V);
// To construct volunteer groups
if (V_i ∈ V_A) then  // if V_i (one of the classified volunteers) belongs to V_A
  if (V_i.C_v ≥ ϑ) then  // if C_v of V_i is greater than or equal to ϑ
    V_i → VG_A';  // V_i is assigned to VG_A'. (→: assign)
  else
    V_i → VG_C';  // V_i is assigned to VG_C'.
  fi;
else if (V_i ∈ V_B) then  // if V_i belongs to V_B
  if (V_i.C_v ≥ ϑ) then  // if C_v of V_i is greater than or equal to ϑ
    V_i → VG_B';  // V_i is assigned to VG_B'.
  else
    V_i → VG_D';  // V_i is assigned to VG_D'.
  fi;
else if (V_i ∈ V_C) then  // if V_i belongs to V_C
  V_i → VG_C';  // V_i is assigned to VG_C'.
else
  V_i → VG_D';  // V_i is assigned to VG_D'.
fi;
```

Fig. 4. Algorithm of volunteer group construction.

Definition 4 (*Volunteer Credibility*). Volunteer credibility C_v represents the probability of result correctness that a volunteer returns.

$$C_v = \frac{CR}{ER + CR + IR}.$$

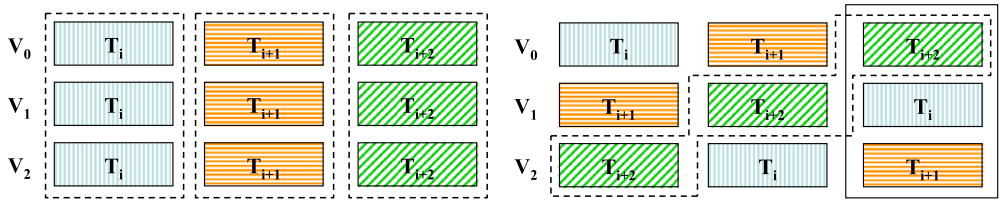
Here, *ER* means the number of erroneous results, *CR* means the number of correct results, and *IR* means the number of incomplete results. $ER + CR + IR$ means the total number of tasks that a volunteer executes. When a volunteer does not complete spot-checking or majority voting on account of crash failure and volunteer autonomy failures, *IR* occurs in a Desktop Grid computing environment. If a volunteer passes the spot-checking, the credibility increases. If volunteers within a voting group reach agreement for majority voting, their credibility also increases.

Volunteer groups are categorized into four classes, as shown in Fig. 3(b), when both Θ and C_v are considered in a grouping procedure. Here, Δ is the expected computation time of a task. ϑ is the desired credibility threshold that a task achieves.

Fig. 4 shows the algorithm of volunteer group construction. Registered volunteers are classified into *A*, *B*, *C*, and *D* classes according to volunteering service time and volunteer availability. Then, each volunteer group is constructed according to volunteer credibility and volunteering service time.

3.1.3. Maintaining volunteer groups

Volunteer groups are constructed at a server through either of two modes: time-based and count-based. The time-based mode builds volunteer groups at regular intervals if the tasks requiring scheduling remain. The count-based mode constructs volunteers groups when the number of participating volunteers is larger than or equal to a predefined number *k*, which depends on the size



(a) Parallel majority voting. (b) Sequential majority voting.

Fig. 5. Construction of voting group.

of volunteer groups or the number of redundancies. The size of a volunteer group s is related with maintenance cost, i.e., the management of task agents and fault tolerance. Volunteer groups are kept until a scheduler can distribute tasks to members. In other words, if all members do not have enough time to execute a task, the volunteer groups are dismissed.

A member of a volunteer group is replaced by another if it continuously returns incorrect results or fails. In the case of failure, the failed member is replaced by a new volunteer.

3.2. Applying scheduling and result certification to volunteer group

3.2.1. Result certification for volunteer group

Result certification is dynamically applied to each volunteer group. GARCM provides the following result certification strategies as follows.

The A' volunteer group has a sufficiently high C_v, high Θ, and high α_v to execute tasks reliably. There is high possibility that the A' volunteer group will produce the correct results. If voting is used for result certification, the sequential voting group approach is more appropriate than the parallel one because the former can perform more tasks. In Fig. 5(b), in case of the T_{i+2} task, if the first two results generated at V_1 and V_0 are the same, there is no need to execute the T_{i+2} task at V_2 because the majority (i.e., 2 out of 3) is already is achieved. Therefore, since other tasks can be executed instead of the executions such as the solid line in Fig. 5(b), the sequential voting group can perform more tasks.

The B' volunteer group has a high C_v and high α_v, but a low Θ. It has a high possibility of producing correct results. However, it cannot complete tasks because of the lack of computation time. In addition, volunteer autonomy failures occur frequently in the middle of execution. Therefore, the manager of B' volunteer group must provide not only task migration in order to execute the tasks continuously but also fault tolerant algorithms to tolerate volunteer autonomy failures. During task migration, the former volunteer affects the latter volunteer (i.e., the volunteer to which a task migrates) to which a task is migrated. In other words, if the latter volunteer is selected wrongly, it might ruin the correct result that was generated by the former volunteer. Therefore, the latter volunteer must be chosen among B' or A' volunteer groups, rather than C' or D' volunteer groups. Spot-checking is additionally performed by the former volunteer as well as by the latter volunteer to check the correctness again. Besides, sequential voting is more appropriate than parallel voting, similar to the A' volunteer group.

The C' volunteer group has a high Θ, but a low C_v and low α_v. It has enough time to execute tasks. However, its results might be incorrect. Therefore, in order to strengthen the credibility, the C' volunteer group must do more spot-checking or place more redundancy than the A' or B' volunteer groups. Parallel voting is more appropriate than sequential voting. If sequential voting is adopted, the voting procedure is frequently delayed because each volunteer suffers from volunteer autonomy failures owing to a low α_v. It also takes longer time and incurs higher overhead to complete result certification. In the case of the parallel voting group, however, the overhead and the completion time are relatively small because the voting procedure for each task is completed within one step, as shown in Fig. 5(a).

The D' volunteer group has a low C_v, low Θ, and low α_v. It has insufficient time to execute tasks. In addition, there is scarcely any possibility of producing correct results. Moreover, volunteer autonomy failures occur frequently in the middle of an execution. Therefore, tasks are not allocated to the D' volunteer group not only because management cost is too expensive, but also because results are incorrect.

3.2.2. Dynamic scheduling for result certification

Each volunteer group has its own scheduling algorithm for result certification according to the above strategies. In general, the tasks are scheduled in the following order, that is, the A', C' and B' volunteer groups sequentially because the A' and C' volunteer groups have enough time to execute tasks. The scheduling algorithms for each volunteer group are as follows.

Scheduling for result certification in the A' volunteer group is as follows. (1) Order the A' volunteer group by α_v and then by Θ. (2) Evaluate the number of redundancies or the spot-checking rate. (3) Construct a sequential voting group, or choose some volunteers for spot-checking on the basis of Θ. (4) Distribute tasks in the manner of a sequential voting group, or allocate special tasks for spot-checking. (5) Check the collected results.

Scheduling for result certification in the B' volunteer group is as follows. (1) Order the B' volunteer group by Θ and then by α_v. (2) The same as for the B' volunteer group. (3) Construct a sequential voting group, or choose some volunteers for spot-checking on the basis of Θ. (4)–(5) The same as for the A' volunteer group. Especially, the B' scheduling algorithm is required to perform additional spot-checking during task migration because of lack of volunteering service time.

Scheduling for result certification in the C' volunteer group is as follows. (1) Order the C' volunteer group by C_v and then α_v. (2) Evaluate the number of redundancies or the spot-checking rate. (3) Construct a parallel voting group, or choose some volunteers for spot-checking on the basis of C_v. (4)–(5) The same as for the A' volunteer group. The C' scheduling algorithm should handle volunteer autonomy failures.

3.2.3. Calculating redundancy and the spot-checking rate

The number of redundancies for the voting and the spot-checking rate are differently applied to each volunteer group. In the case of redundancy for voting, the C' volunteer group has a greater redundancy number than the A' and B' volunteer groups because of a low credibility. Similarly, spot-checking rate is ordered as C', B' and A'. The B' volunteer group has a higher number of spot-checking than A' volunteer group on account of task migration.

The number of redundancies r for majority voting is dynamically calculated through Eq. (1). Here, $r = 2k + 1$. The final error rate of majority voting is evaluated as follows [35].

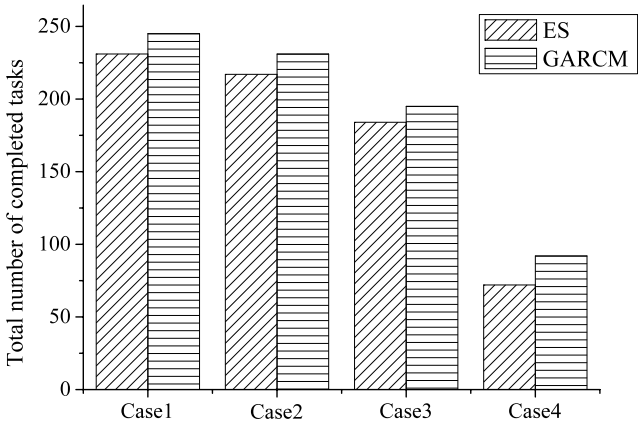


Fig. 6. The number of completed tasks without result certification.

$$\varepsilon(C_v', r) = \sum_{i=k+1}^{2k+1} \binom{2k+1}{i} (1 - C_v')^i (C_v')^{(2k+1-i)} \quad (1)$$

which is bounded by $\frac{[4C_v'(1-C_v')]^{k+1}}{2(2C_v'-1)\sqrt{\pi k}}$.

Here, parameter C_v' means the probability that volunteers within each volunteer group generate correct results.

Suppose that a desired credibility threshold is ϑ. GARCM calculates the number of redundancies for each volunteer group if $(1 - \vartheta) \geq \varepsilon(C_v', r)$. Consequently, the A' and B' volunteer groups have a small r, so it can reduce the overhead of majority voting and execute more tasks. In contrast, the C' volunteer group has a large r. The large r makes the credibility high.

Spot-checking rate q is also calculated through Eq. (2). The final error rate of spot-checking is evaluated as follows [30].

$$\varepsilon(q, n, C_v', s) = \frac{sC_v'(1 - qs)^n}{(1 - C_v') + C_v'(1 - qs)^n} \quad (2)$$

where, n is the saboteur's share in the total work and s is its sabotage rate.

If n and s are given, the spot-checking rate, q, of each volunteer group is calculated. GARCM calculates the rate of spot-checking for each volunteer group if $(1 - \vartheta) \geq \varepsilon(q, n, C_v', s)$. The spot-checking rates of the A' and B' volunteer groups are smaller than that of the C' volunteer group. Therefore, the A' and B' volunteer groups can reduce overhead, and therefore execute more tasks. The C' volunteer group can increase its credibility.

4. Analysis and evaluation

4.1. Analysis

Existing result certification mechanisms are analyzed according to the taxonomy shown in Fig. 2. Table 1 shows a mapping of the taxonomy to existing result certification mechanisms.

According to survey and analysis, result certification should be tightly related to scheduling in the sense that both the special task for spot-checking and the redundant tasks for voting are allocated to volunteers in a scheduling procedure. However, most existing Desktop Grid systems simply use eager scheduling, not considering volunteer reputation. As a result, there are high overhead, performance degradation, and scalability problems. Desktop Grid systems should provide a new scheduling mechanism for result certification, coupling resource reputation that accounts for volatility, volunteering time and credibility with result certification.

Result certification should be coupled with an incentive mechanism. Incentive mechanisms aim to encourage resource owners to donate resources eagerly, reliably and trustworthily by giving rewards (e.g., money, resources, credit, ranking, etc.) to volunteers for their donation, or penalties (e.g., blacklist) to volatile and malicious volunteers. However, most existing result certification mechanisms do not give incentive to reliable volunteers in scheduling and result certification procedures. To make Desktop Grid systems more reliable, Desktop Grid systems should consider incentive mechanisms.

4.2. Simulation results

We evaluate GARCM with existing scheduling mechanism for result certification. This evaluation focuses on how much performance improvement will be gained depending on whether or not the volunteer groups are considered in scheduling and result certification procedures. GARCM is compared to eager scheduling to which result certification has been applied. There are considerable scheduling heuristics in Grids, e.g., MCT, MET, SA, KPB, min–min, max–min, and sufferage heuristics [36,37]. In this article, we adopt eager scheduling among existing scheduling heuristics because it is more straightforward and simple than other heuristics in Grids. In particular, eager scheduling has been mainly used in Desktop Grids [19,32,20,33,21] because it is more adaptive to the dynamic computing environments of Desktop Grids than the heuristics in Grids.

In our simulations, we model a number of scenarios using different distributions for volunteer properties, as shown in Table 2. The evaluation is based on a simulator modeled on Korea@Home [34], which consists of a server and many volunteers. We intentionally set up volunteer groups which have different volunteering service time Θ and volunteer availability α_v. As the performance metrics for comparing GARCM to eager scheduling, our simulations use the number of completed tasks, the number of redundancy and the error rate.

Table 2 shows a simulation environment with different volunteer groups, volunteering service time, and volunteer availability. For each case in Table 2, 200 volunteers participated in our simulation for one hour. In Case 1, the A' volunteer group has more volunteers than the other groups. In Case 2, the C' volunteer group has more volunteers than the other groups. Case 3 shows that more volunteers belong to the A' and C' volunteer groups as compared to the other groups. In Case 4, the D' volunteer group has more volunteers than the other groups. When analyzing Table 2, Case 1 is different from Case 2 with respect to volunteer credibility. Case 3 is different from Case 1 with regard to volunteer availability and volunteer availability. Case 4 is different from Case 1 with respect to volunteer availability and volunteering service time. Here, A', B', C', and D' represent the A', B', C', and D' volunteer groups, respectively. Here, P. (i.e., population) represents the number of volunteers. Each simulation was repeated 10 times per each case.

The range of *MVT* is set as 10–60 min. *MTTVAF* is configured as 1/0.2–1/0.05 min. *MTTR* is set as 3–10 min. A task in the application exhibits 18 min of execution time on a dedicated Pentium 1.4 GHz. Suppose that $s = 0.1$ and $n = 10$ in spot-checking.

Figs. 6–8 show the simulation results. Here, the ES represents an existing eager scheduling mechanism. GARCM(A'), GARCM(B') and GARCM(C') mean the results performed by each volunteer group, respectively. As shown in Figs. 6 and 7(a), and 8(a), GARCM completes more tasks than the existing eager scheduling mechanism, while satisfying the desired credibility threshold (or desired error rate), as shown in Figs. 7(c), and 8(c). In particular, the A' volunteer group has an important role in gaining better performance. When the number of members in the A' volunteer group decreases gradually (i.e., from Case 1 to Case 4), the number of completed tasks decreases. In contrast, as the number

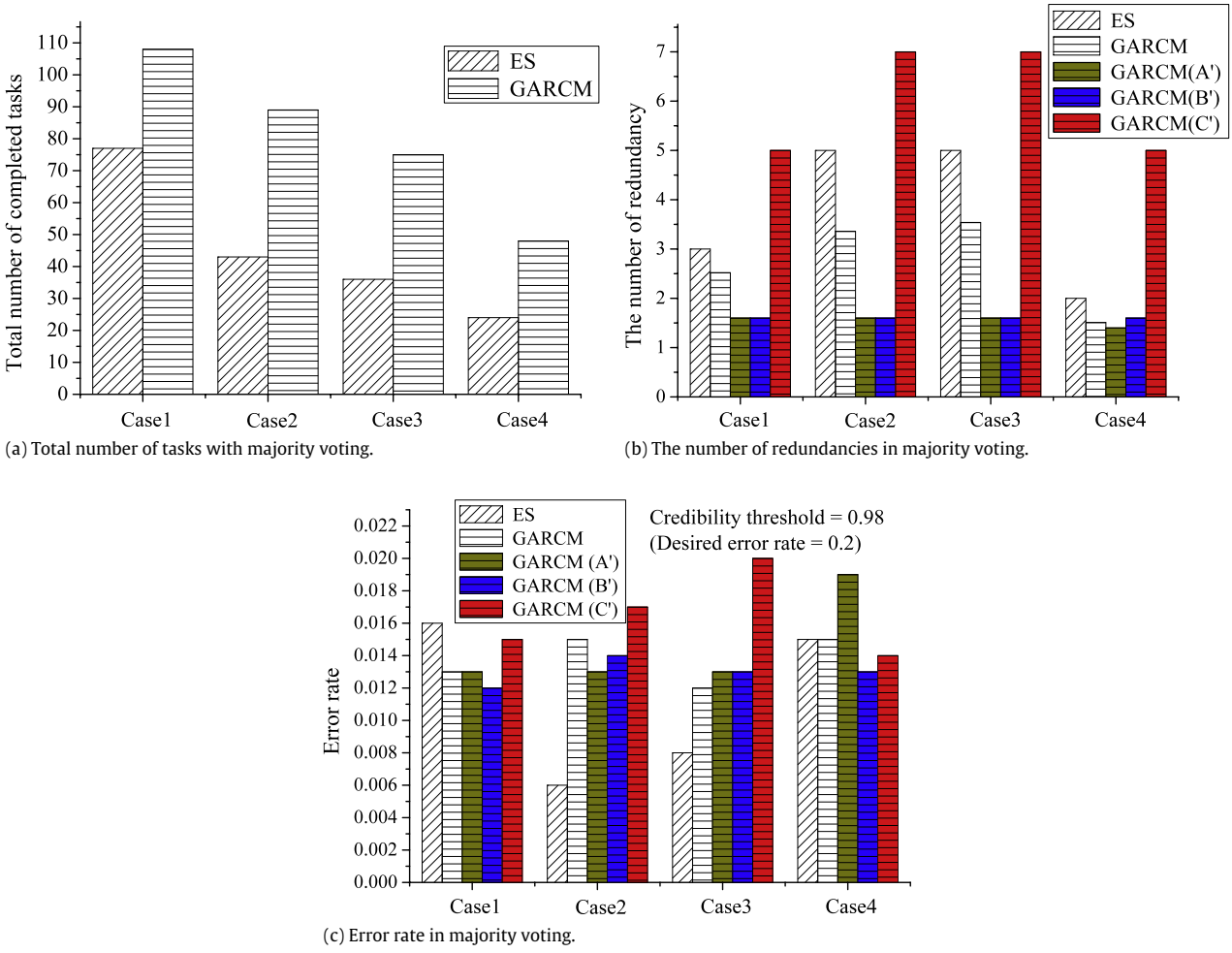
Table 1
Analysis of result certification.

Mechanism	Comparison object	Judgment method	Resource selection method	Calculation method	Response	Description
Golle et al. [1]	Result of a special task (one-way hash function)	Sampling (simple sampling–randomly selected input)	Not mentioned	Not mentioned	Not mentioned	-Propose ringers and bogus ringers mechanisms
Sarmenta [2]	-Result of a special task for sampling -Final result of a task for voting	-Sampling (threshold-based weighted sampling) -Voting (majority voting, *m*-first voting, threshold-based weighted voting)	FCFS (eager scheduling)	-Reputation (failure rate, sabotage rate) -Threshold (acceptable error rate)	-Penalty (blacklist)	-Credibility-based enhanced eager scheduling
Germain-Renaud et al. [3,4]	Result of a special task	Sampling (threshold-based sampling)	Not mentioned	Threshold (acceptable error rate)	-Penalty (blacklist)	-Three scenarios: Normal, massive attack, subtle attack -Fault tolerant application (Monte-Carlo simulations)
Du et al. [5]	Result of a special task (Merkle tree with one-way hash function)	Sampling (simple sampling–randomly selected input)	Not mentioned	Not mentioned	Not mentioned	-Merkle-tree based commitment technique
Varrette et al. [6], Krings et al. [7]	Final result of a task	Sampling (simple sampling, Threshold-based sampling)	Not mentioned	Threshold (acceptable error rate)	Not mentioned	-Result certification for tasks with dependency -Formalization of result certification
Yang et al. [8]	Result of a special task (R-beacon)	Sampling (simple sampling)	Not mentioned	Not mentioned	Not mentioned	-Insert R-beacon into a significant computation region -R-beacon message initiates result certification
Taufer et al. [9]	Final result of a task	-Voting (majority voting)	Qos (preference-homogeneous HW/SW)	Non-reputation (odd number)	Not mentioned	-Homogeneous redundancy
Zhao et al. [10]	-Result of a special task for sampling (quiz) -Final result of a task for voting	-Sampling (simple sampling) -Voting (majority voting)	Reputation (trust)	Reputation (failure rate and sabotage rate)	-Penalty (decrease the trust value, or blacklist) -Incentive (increase the trust value)	-Coupling result certification with reputation systems
Sonnek et al. [11]	Final result of a task	-Voting (majority voting, threshold-based weighted voting)	Reputation (reliability)	Reputation (reliability) Threshold (LOC: likelihood of correctness)	Not mentioned	-Adaptive reputation-based scheduling
Domingues et al. [12], Kondo et al. [13]	Intermediate result of a task (checkpoint)	Voting (majority voting)	Not mentioned	Not mentioned	Not mentioned	-Long running applications (climatedprediction.net, climatechange projects)
Kim et al. [14]	Final result of a task	Voting (majority voting)	-Reputation (credibility, result return rate) -Qos (deadline)	Not mentioned	Not mentioned	-Trust-based sabotage tolerance for deadline tasks
Proposed mechanism	-Result of a special task for sampling -Final result of a task for voting	-Sampling (Threshold-based sampling) -Voting (majority voting, threshold-based weighted voting)	Reputation (volatility, credibility, volunteering service time)	-Reputation (failure rate, sabotage rate, volatility, credibility) -Threshold (acceptable error rate)	-Penalty (exclusion in a scheduling procedure)	-Group-based result certification

of members in the D' volunteer group increases, the number of completed tasks decreases. In addition, volunteer availability is tightly related to performance. Cases 1 and 2 can complete more tasks than cases 3 and 4.

In the case of majority voting, GARCM obtains more task results than eager scheduling because it dynamically decides the number of redundancies according to properties of volunteer groups, as shown in Fig. 7(b). The A' and B' volunteer groups choose less redundancy than the C' volunteer group. As a result, the A' and B' volunteer groups are able to reduce replication overhead, so they can execute more tasks. On the other hand, the C' volunteer group needs a large number of redundancies to meet the reliability threshold. However, it can decrease its error rate.

In the case of spot-checking, GARCM completes more tasks than eager scheduling because it dynamically decides the spot-checking rate according to the properties of the volunteer groups, as shown in Fig. 8(b). However, if the A' volunteer group is less than the C' volunteer group, as in Cases 2 and 3, the number of completed tasks



(a) Total number of tasks with majority voting.

(b) The number of redundancies in majority voting.

(c) Error rate in majority voting.

Fig. 7. Simulation results in group-based voting.

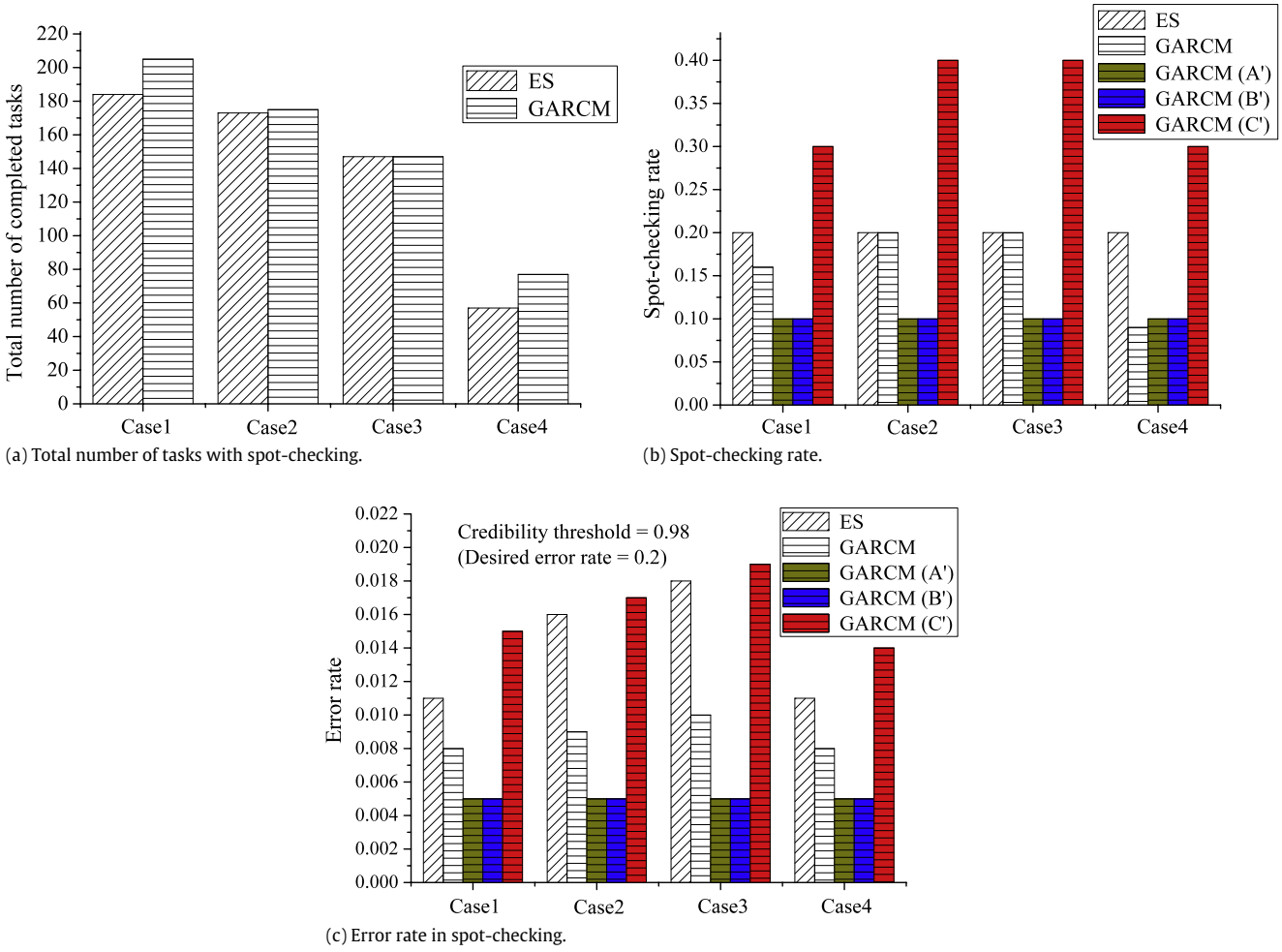
Table 2
Simulation environment.

Case		A'	B'	C'	D'	Total
Case 1	P_v	84 (42%)	26 (13%)	70 (35%)	20 (10%)	200
	α_v	0.84	0.88	0.81	0.83	0.84
	Θ	41	17	39	16	35 min
	C_v	0.98	0.98	0.88	0.86	0.93
Case 2	P_v	71 (35.5%)	31 (15.5%)	76 (38%)	22 (11%)	200
	α_v	0.87	0.89	0.80	0.82	0.84
	Θ	41	17	39	16	34 min
	C_v	0.98	0.98	0.84	0.85	0.91
Case 3	P_v	76 (38%)	27 (13.5%)	80 (40%)	17 (8.5%)	200
	α_v	0.86	0.78	0.80	0.71	0.81
	Θ	35	17	33	16	30 min
	C_v	0.98	0.98	0.82	0.85	0.91
Case 4	P_v	42 (21%)	59 (29.5%)	30 (15%)	69 (34.5%)	200
	α_v	0.80	0.70	0.78	0.69	0.73
	Θ	28	12	25	13	24 min
	C_v	0.98	0.98	0.89	0.89	0.94

becomes similar because the C' volunteer group has a high spot-checking rate.

5. Conclusions

Result certification is important to guarantee reliable execution. A dynamic scheduling mechanism for result certification is also essential to overcome high overhead and performance degradation. In this article, we proposed a new Group-based Adaptive Result Certification Mechanism (GARCM). GARCM reflects the volatility, failure, heterogeneous properties and reliability of volunteers in scheduling and result certification procedures by means of volunteer groups. Volunteer groups are constructed according to volunteer properties such as availability, volunteering service time and credibility. GARCM dynamically applies different scheduling and result certification algorithms to each volunteer group.



(a) Total number of tasks with spot-checking.

(b) Spot-checking rate.

(c) Error rate in spot-checking.

Fig. 8. Simulation results in group-based spot-checking.

The simulation results showed that GARCM gains better performance and reduces the overhead of result certification while satisfying the desired credibility threshold. In particular, GARCM completes more tasks than eager scheduling to which result certification is applied result certification. GARCM reduces the number of redundancies or the spot-checking rate more than eager scheduling because it dynamically decides the number of redundancies or spot-checking rate according to the properties of each individual volunteer group. With regard to volunteer groups, the evaluation results showed that the larger the number of volunteers in the A' and C' volunteer groups is, the larger the number of completed tasks. Also, as the number of volunteers in the B' and D' volunteer groups increases, the more the difference between GARCM and eager scheduling.

We are currently studying how to couple result certification with incentive scheduling in Desktop Grids. An incentive scheduling is aimed at giving more rewards and benefit to eager, reliable and trustworthy volunteers, and punishing volatile, selfish or malicious volunteers. Thus, it encourages volunteers to donate their resources eagerly and reliably. Incentive scheduling can use result certification to evaluate volunteers.

Acknowledgments

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2007-357-D00207) and the Department of Computer Science and Software Engineering, University of Melbourne. We are grateful to Peter Sylvestre for sharing their thoughts on the topic. We thank the anonymous referees of this article for their valuable comments.

References

[1] P. Golle, I. Mironov, Uncheatable distributed computations, in: Proc. of the 2001 Conf. on Topics in Cryptology: The Cryptographer's Track at RSA, CT-RSA 2001, Apr. 2001, pp. 425–440.
[2] L.F.G. Sarmenta, Sabotage-tolerance mechanisms for volunteer computing systems, Future Generation Computer Systems 18 (4) (2002) 561–572.
[3] C. Germain-Renaud, N. Playez, Result checking in global computing systems, in: Proc. of the 17th Annual ACM Intl. Conf. on Supercomputing, ICS'03, Jun. 2003, pp. 226–233.
[4] C. Germain-Renaud, D. Monnier-Ragaigne, Grid result checking, in: Proc. of the 2nd conference on Computing Frontiers, CT'05, May. 2005, pp. 87–96.
[5] W. Du, J. Jia, M. Mangal, M. Murugesan, Uncheatable grid computing, in: Proc. of the 24th IEEE Intl. Conf. on Distributed Computing Systems, ICDCS'04, Mar. 2004, pp. 4–11.
[6] S. Varrette, J. Roch, F. Leprevost, FlowCert: Probabilistic certification for peer-to-peer computations, in: Proc. of the 16th Symposium on Computer Architecture and High Performance Computing, SBAC-PAD'04, Oct. 2004, pp. 108–115.
[7] A. Krings, J. Roch, S. Jafar, S. Varrette, A probabilistic approach for task and result certification of large-scale distributed applications in hostile environments, in: Advances in Grid Computing — European Grid Conference, EGC 2005, in: LNCS, vol. 3470, 2005, pp. 323–333.
[8] S. Yang, A.R. Butt, Y.C. Hu, S.P. Midkiff, Trust but verify: Monitoring remotely executing programs for progress and correctness, in: Proc. of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'05, Jun. 2005, pp. 196–205.
[9] M. Taufer, D. Anderson, P. Cicotti, C.L. Brooks III, Homogeneous redundancy: A technique to ensure integrity of molecular simulation results using public computing, in: Proc. of the 19th IEEE Intl. Parallel and Distributed Processing Symposium, IPDPS'05, Heterogeneous Computing Workshop, HCW'05, Apr. 2005, pp. 119a.

[10] S. Zhao, V. Lo, Result verification and trust-based scheduling in open peer-to-peer cycle sharing systems, in: Proc. of the Fifth IEEE Intl. Conf. on Peer-to-Peer Computing, P2P'05, Sept. 2005, pp. 31–38.
[11] J. Sonnek, A. Chandra, J. Weissman, Adaptive reputation-based scheduling on unreliable distributed infrastructures, IEEE Transactions on Parallel and Distributed Systems 18 (11) (2007) 1551–1564.
[12] P. Dominguesa, B. Sousab, L.M. Silva, Sabotage-tolerance and trust management in desktop grid computing, Future Generation Computer Systems 23 (7) (2007) 904–912.
[13] D. Kondo, F. Araujo, P. Domingues, L.M. Silva, Result error detection on heterogeneous and volatile resources via intermediate checkpointing, in: Proc. of CoreGRID Integration Workshop 2007, Jun. 2007.
[14] H. Kim, C. Hwang, S. Lee, S. Choi, J. Gil, Priority based list scheduling for sabotage-tolerance with deadline tasks in desktop grids, International Journal of Computer Systems Science & Engineering 23 (2) (2008) 77–87.
[15] F. Berman, G.C. Fox, A.J.G. Hey, Grid Computing: Making the Global Infrastructure a Reality, Wiley, 2003.
[16] M. Baker, R. Buyya, D. Laforenza, Grids and Grid technologies for wide-area distributed computing, Software: Practice and Experience 32 (15) (2002) 1437–1466.
[17] I. Foster, A. Iamnitchi, On death, taxes, and the convergence of peer-to-peer and grid computing, in: Proc. of the 2nd Intl. Workshop on Peer-to-Peer Systems, IPTPS'03, Feb. 2003.
[18] D.S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu, Peer-to-peer computing, HP Laboratories Palo Alto HPL-2002-57, Mar. 2002.
[19] D.P. Anderson, BOINC: A system for public-resource computing and storage, The Fifth IEEE/ACM Intl. Workshop on Grid Computing, GRID'04, Nov. 2004, pp. 4–10.
[20] L.F.G. Sarmenta, S. Hirano, Bayanihan: Building and studying web-based volunteer computing systems using Java, Future Generation Computer Systems, Special Issue on Metacomputing 15 (5–6) (1999).
[21] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Neri, O. Lodygensky, Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid, Future Generation Computer Systems 21 (3) (2005) 417–437.
[22] A. Chien, B. Calder, S. Elbert, K. Bhatia, Entropia: Architecture and performance of an enterprise desktop grid system, Journal of Parallel and Distributed Computing 63 (5) (2003) 597–610.
[23] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: The condor experience, Concurrency and Computation: Practice and Experience 17 (2–4) (2005) 323–356.
[24] D. Kondo, M. Taufer, J. Karanicolas, C.L. Brooks, H. Casanova, A. Chien, Characterizing and evaluating desktop grids: An empirical study, in: Proc. of the 18th Intl. Parallel and Distributed Processing Symposium, IPDPS 2004, Apr. 2004, pp. 26–35.
[25] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, C. Hwang, Characterizing and classifying desktop Grid, in: Proc. of IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGRID 2007), Sixth Intl. Workshop on Global and Peer to Peer Computing, GP2P, May 2007, pp. 743–748.
[26] The Great Internet Mersenne Prime Search (GIMPS), http://www.mersenne.org/.
[27] Distributed.net, http://distributed.net.
[28] SETI@home, http://setiathome.ssl.berkeley.edu.
[29] S. Choi, M. Baik, C. Hwang, J. Gil, H. Yu, Volunteer availability based fault tolerant scheduling mechanism in desktop grid computing environment, in: Proc. of the 3rd IEEE Intl. Symposium on Network Computing and Applications, Workshop on Adaptive Grid Computing, NCA-AGC2004, Aug. 2004, pp. 476–483.
[30] H. Wasserman, M. Blum, Software reliability via run-time result-checking, Journal of the ACM (JACM) 44 (6) (1997) 826–849.
[31] S. Choi, M. Baik, J. Gil, C. Park, S. Jung, C. Hwang, Dynamic scheduling mechanism for result certification in peer to peer grid computing, in: Proc. of the Intl. Conf. on Grid and Cooperative Computing, GCC 2005, in: LNCS, vol. 3795, 2005, pp. 811–824.
[32] A. Baratloo, M. Karaul, Z.M. Kedem, P. Wijckoff, Charlotte: Metacomputing on the web, Future Generation Computer Systems, Special Issue on Metacomputing 15 (5–6) (1999) 559–570.
[33] M.O. Neary, P. Cappello, Advanced eager scheduling for Java-based adaptive parallel computing, Concurrency and Computation: Practice and Experience 17 (7–8) (2005) 797–819.
[34] Korea@Home, http://www.koreaathome.org/eng/.
[35] Y.A. Zuev, On the estimation of efficiency of voting procedures, Theory of Probability & its Applications 42 (1) (1998) 73–81.
[36] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems, in: Proc. of the 8th Heterogeneous Computing Workshop, HCW'99, Apr. 1999, pp. 30–44.
[37] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, R.F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810–837.

SungJin Choi is a Post-doctoral researcher in the Department of Computer Science and Software Engineering at University of Melbourne. His research interests include Desktop Grid computing, Grid computing, mobile agent, peer-to-peer computing, and distributed systems.

Mr. Choi received a Ph.D. in computer science from Korea University. He is a member of the IEEE and ACM. Contact him at lotieye@gmail.com or lotieye@disys.korea.ac.kr.

Rajkumar Buyya is a professor in the Department of Computer Science and Software Engineering at University of Melbourne. His research interests include Grid computing, Cloud computing and distributed systems.

Dr. Buyya received a Ph.D. in computer science and software engineering from Monash University. Contact him at raj@csse.unimelb.edu.au.