A Peer to Peer Grid Computing System Based on Mobile Agents

Joon-Min Gil[1] and Sung-Jin Choi[2]

[1] Department of Computer Science Education, Catholic University of Daegu
330 Geumnak, Hayang-eup, Gyeongsan-si, Gyeongbuk 712-701, Korea
jmgil@cu.ac.kr

[2] Department of Computer Science and Engineering, Korea University
5-1 Anam-dong, Sungbuk-ku, Seoul 136-701, Korea
lotieye@disys.korea.ac.kr

Abstract. In a peer to peer grid computing system, volunteers (i.e., resource provides) with heterogeneous properties can freely join and leave in the middle of their computation. Thus, the system should be adaptive to a dynamic changing environment. In particular, scheduling, result certification, and replication mechanisms must be dynamic and adaptive in such a system. In this paper, we propose a new peer to peer grid computing system based on mobile agents. The proposed system constructs volunteer groups according to volunteers' dynamic properties such as service time, availability, and credibility. For each volunteer groups, different scheduling, result certification, replication mechanisms are used. These mechanisms are implemented as mobile agents and are conducted in a decentralized way.

1 Introduction

A peer to peer grid computing system is a platform that achieves a high throughput computing by harvesting a number of idle desktop computers owned by individuals (i.e., volunteers) at the edge of the Internet using peer to peer computing technologies [1,2]. It usually supports embarrassingly parallel applications that consist of a lot of instances of the same computation with each own data.

A peer to peer grid computing is complicated by heterogeneous capabilities, failures, volatility (i.e., intermittent presence), and lack of trust [3,4]. The volunteers that are based on desktop computers at the edge of Internet, have various capabilities (i.e., CPU, memory, network bandwidth, and latency), and are exposed to link and crash failures. Moreover, they are free to join and leave in the middle of execution without any constraints. Accordingly, they have various volunteering times, and *public execution* (i.e., the execution of a task as a volunteer) can be stopped arbitrarily on account of unexpected leave. Since volunteers are not totally dedicated to a peer to peer grid computing system, the public execution can be temporarily suspended by *private execution* (i.e., the execution of a private job as a personal user). These unstable situations lead to the delay and blocking of the execution of tasks. This paper regards these situations as *volunteer autonomy failures.* Volunteers have different occurrence rates for volunteer autonomy failures according to their execution behaviors. Moreover, a peer to peer grid computing system suffers from the corrupted results executed by malicious volunteers. This is due to the voluntary participation of volunteers without any constraints. Consequently, the system must detect and tolerate the erroneous results to guarantee reliable execution from such an untrustworthy environment. These distinct features make it difficult for a volunteer server to schedule tasks and manage volunteers.

In order to improve the reliability of computation and gain better performance, the peer to peer grid computing system should adapt to dynamic environment. However, existing systems do not provide adaptive and dynamic scheduling, result certification, and replication mechanisms per group basis. In addition, their mechanisms are performed only by the volunteer server in a centralized way. As a result, existing systems have high overhead and deteriorate overall performance. To solve the problems, we propose a new peer to peer grid computing system based on mobile agents. The proposed system applies different scheduling, result certification, replication algorithms to the volunteer groups that are classified on the basis of their properties such as volunteering service time, availability, and credibility; the different algorithms are implemented as mobile agents and are conducted in a decentralized way.

This paper organized as follows. Section 2 presents why mobile agents are used and describes our execution model. Section 3 presents a peer to peer grid computing system based on mobile agents in detail. In Section 4, implementation and evaluation for our mechanism will be presented. Finally, our conclusion is given in Section 5.

2 System Model

2.1 Why Mobile Agent?

Mobile agent technology [5] is exploited to make the scheduling mechanism adaptive to dynamically changing peer to peer grid computing environments. There are some advantages of making use of mobile agents in the environments.

1) *Various scheduling mechanisms can be performed at a time according to the properties of volunteers.* For example, these scheduling mechanisms can be implemented as mobile agents (i.e., scheduling mobile agents). After volunteers are classified into volunteer groups, the most suitable scheduling mobile agent for a specific volunteer group is assigned to the volunteer group according to its property. Existing peer to peer grid computing systems, however, cannot apply various scheduling mechanisms because only one scheduling mechanism is performed by a volunteer server in a centralized way.
2) *A mobile agent can decrease the overhead of volunteer server by performing scheduling, result certification, and replication algorithms in a decentralized way.* The scheduling mobile agents are distributed to volunteer groups. Then, they autonomously conduct scheduling, fault tolerance, and replication algorithms in

each volunteer group without any direct control of a volunteer server. Accordingly, the volunteer server does not further undergo the overhead.
3) *A mobile agent can adapt to dynamically changing peer to peer grid computing environments.* In a peer to peer grid computing environment, volunteers can join and leave at any time. In addition, they are characterized by heterogeneous properties such as capabilities (i.e., CPU, storage, or network bandwidth), location, availability, credibility, and so on. These environmental properties are changing over time. A mobile agent can perform asynchronously and autonomously, while coping with these changes.

2.2 Execution Model

Fig. 1 illustrates the execution model based on mobile agents in peer to peer grid computing environments. In the registration phase, volunteers register basic properties such as CPU, memory, OS type as well as additional properties including *volunteering time*, *volunteering service time*, *volunteer availability*, *volunteer autonomy failures*, *volunteer credibility*, and so on. Since these additional properties are related to dynamical execution, they are more important than basic properties. In the job submission phase, the submitted job is divided into a number of tasks. The tasks are implemented as mobile agents (i.e., task mobile agents: T-MA). In the task allocation phase, the volunteer server does not perform entire scheduling anymore. Instead, it helps scheduling mobile agents (S-MA) to perform a scheduling procedure. Initially, the volunteer server classifies and constructs the volunteer groups according to properties such as location, volunteer autonomy failures, volunteering service time, and volunteer availability. Next, scheduling mobile agents are distributed to volunteer groups according to their properties. Finally, each scheduling mobile agent distributes task mobile agents to the members of its volunteer group. In the task execution phase, the task mobile agent is executed in cooperation with its scheduling mobile agent while migrating to another volunteer or replicating itself in the presence of failures. In the task result return phase, the task mobile agent returns each result

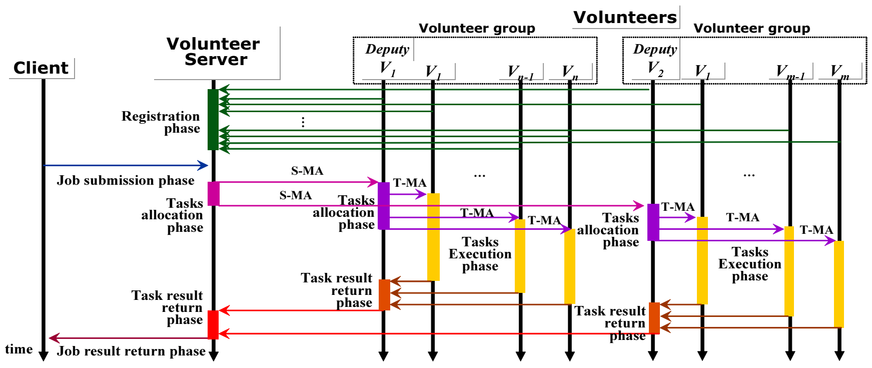


Fig. 1. Execution model based on mobile agents

to its scheduling mobile agent. When all task mobile agents return their results, the scheduling mobile agent aggregates the results and then returns the collected results to the volunteer server. In order to tolerate erroneous results, majority voting and spot-checking mechanisms are conducted. In the job result return phase, the volunteer server returns a final result to the client when it receives all the results from the scheduling mobile agents.

The main differences between our model and existing ones are as follows: 1) The new kinds of mobile agents are considered as the scheduling and task mobile agents. 2) They use the volunteer groups that are constructed according to dynamic properties of volunteers such as autonomy failures, service time, availability, and credibility. 3) Various scheduling, result certification, and replication algorithms are performed simultaneously in a decentralized way. In fact, there has been the use of mobile agents in the literature [6]. However, the migration of mobile agents in master-worker model is mainly considered.

3 Peer to Peer Grid Computing System Using Mobile Agents

This section describes a peer to peer grid computing system using mobile agents in detail. First, it provides the construction mechanism of volunteer groups. Then, adaptive scheduling, result certification, replication mechanisms are presented.

3.1 Volunteer Group Construction Mechanism

To apply different scheduling and result certification algorithm suitable for volunteers in a scheduling phase, volunteers are required to first be formed into homogeneous groups. Our construction mechanism classifies volunteers into four volunteer groups on the basis of *volunteer availability* α_v, *volunteering service time* Θ, and *volunteer credibility* C_v.

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

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

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

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

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

Here, the $MTTVAF$ (Mean Time To Volunteer Autonomy Failures) means the average time before the volunteer autonomy failures happen, and the $MTTR$ (Mean Time To Rejoin) means the mean duration of volunteer autonomy failures. The α_v reflects the degree of volunteer autonomy failures, whereas the traditional availability in distributed systems is mainly related with the crash failure.

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 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.

Definition 4 (Volunteer credibility). *Volunteer credibility C_v is the probability that represents correctness of the results which a volunteer will produce.*

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

Here, ER, CR, and IR mean the number of erroneous results, the number of correct results, and the number of incomplete results, respectively. The sum of ER, CR, and IR means the total number of tasks that a volunteer executes. The IR occurs when a volunteer does not complete spot-checking or majority voting on account of crash failure and volunteer autonomy failures.

When both Θ and C_v are considered in grouping volunteers, volunteer groups are categorized into four kinds of classes (A', B', C', and D') as shown in Fig. 2. In this figure, Δ and ϑ represent the expected computation time of a task and the desired credibility threshold which a task achieves, respectively.

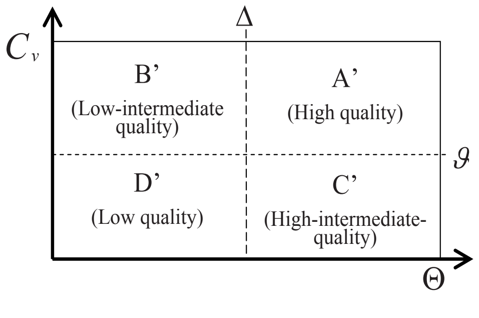


Fig. 2. The classification of volunteer groups

3.2 Group Based Scheduling Mechanism

Differently from existing scheduling mechanisms [7,8,9], our scheduling mechanism is based on volunteer group and mobile agents.

Allocating Scheduling Mobile Agents to Scheduling Groups. After constructing volunteer groups, a volunteer server allocates the scheduling mobile agents (S-MA) to volunteer groups. However, it is not practical to allocate S-MAs directly to the volunteer groups in a scheduling procedure because some volunteer groups are not perfect for finishing tasks reliably. Therefore, we need making new scheduling groups by combining the volunteer groups each other: $A'D'$ & $C'B'$, $A'B'$ & $C'D'$, and $A'C'$ & $B'D'$. In this paper, we consider the first combination in scheduling because B' volunteer group compensates for C' volunteer group with regard to volunteer availability.

The S-MA is executed in the deputy volunteer which is selected among members in A' volunteer group. Accordingly, deputy volunteers have high volunteer availability and volunteering service time. Also, they have enough hard-disk capacity and network bandwidth.

Distributing Task Mobile Agents to Group Members. A task mobile agent (T-MA) consists of a parallel code and data. After S-MAs are allocated to the scheduling groups, each S-MA distributes T-MAs to the members of the scheduling group. The S-MAs perform different scheduling, result certification, and replication algorithms according to the type of volunteer groups.

The S-MA of the $A'D'$ scheduling group performs the scheduling as follows.

1) *Order the A' volunteer group by α_v and then by Θ.* 2) *Distribute T-MAs to the arranged members of the A' volunteer group.* 3) *If a T-MA fails, replicate the failed task to a new volunteer selected in the A' volunteer group by means of the replication algorithm, or migrate the task to a volunteer selected in the A' or B' volunteer groups if task migration is allowed.*

The S-MA of the $C'B'$ scheduling group performs the scheduling as follows.

1)*Order the C' and B' volunteer groups by α_v and then by Θ.* 2) *Distribute T-MAs to the arranged members of the C' volunteer group.* 3) *If a T-MA fails, replicate the failed task to a new volunteer selected in the ordered C' volunteer groups, or migrate the task to a volunteer selected in the B' or C' volunteer groups.*

Tasks are firstly distributed to the $A'D'$ scheduling group and then the $C'B'$ scheduling group. They are also distributed to the volunteers with high α_v and long Θ. In our scheduling, if checkpointing is not used, tasks are not allocated to the B' and D' volunteer groups, because they have insufficient time to finish the task reliably. In this case, the B' and D' volunteer groups execute tasks for testing, i.e., to measure their properties. For example, the tasks executed in the A' and C' volunteer groups are redistributed to the D' and B' volunteer groups, respectively. However, the B' volunteer group can be used to assist the main volunteer groups (i.e., A' or C') if task migration is permitted. The volunteer group B' in the scheduling group $C'B'$ can be used to compensate for the C' volunteer group with regard to volunteer availability. Suppose that a volunteer in the C' volunteer group suffers from volunteer autonomy failures. If the volunteering time of a volunteer in the B' volunteer group implies the duration of volunteer autonomy failures at the failed volunteer, the suspended task can migrate to a new volunteer in the B' volunteer group.

3.3 Group Based Replication Mechanism

The group based replication mechanism automatically adjusts the number of redundancy, and selects an appropriate replica according to the properties of each volunteer group.

How to calculate the number of redundancy. If replication is used, each S-MA calculates the number of redundancy for its volunteer group. It exploits volunteer autonomy failures, volunteer availability, and volunteering service time simultaneously when calculating the number of redundancy.

In a peer to peer grid computing environment, volunteer autonomy failures occur much more frequently than crash and link failures. Moreover, the rates and types of volunteer autonomy failures are various. Accordingly, the number of redundancy must be calculated on the basis of volunteer groups that have similar rate and types of volunteer autonomy failures in order to reduce the replication overhead.

On the assumption that the lifetime of a system is exponentially distributed [7,10], the number of redundancy r for reliability is calculated by

$$\begin{gathered}(1 - e^{-\Delta/\tau'})^r \leq 1 - \gamma \\ \tau' = (V_0 \cdot \tau + V_1 \cdot \tau + \cdots + V_n \cdot \tau)/n \end{gathered} \tag{1}$$

where, τ and τ' represent the MTTVAF of a volunteer and the MTTVAF of a volunteer group, respectively; n is the total number of volunteers within a volunteer group; $V_n \cdot \tau$ means τ of a volunteer V_n; γ is the reliability threshold.

In (1), the term $e^{-\frac{\Delta}{\tau'}}$ represents the reliability of each volunteer group, which means the probability to complete the tasks within Δ. It reflects volunteer autonomy failures. The $(1 - e^{-\frac{\Delta}{\tau'}})^r$ means the probability that all replicas fail to complete the replicated tasks. If the required reliability γ is provided, the value of r is calculated using (1). Each volunteer group has different r; e.g., the volunteer group A' and C' have smaller r than the volunteer group B'.

How to distribute T-MAs to replicas. The methods of distributing tasks to replicas are categorized into two approaches: parallel distribution and sequential distribution. In the parallel distribution (Fig. 3(a)), the task T_m is distributed to all members (V_0, V_1, and V_2), and then executed simultaneously. In the sequential distribution (Fig. 3(b)), the T_m is distributed and executed sequentially.

In the case of the A' volunteer group, sequential distribution is more appropriate than parallel distribution because the former can complete more tasks. For example, in Fig. 3(b), if V_0 completes the task T_m, there is no need to execute it at V_1 and V_2. If the A' volunteer group performs parallel distribution, it exhibits the overhead of replication in the sense that volunteers execute the same tasks even though they are able to execute other tasks. In contrast to the A' volunteer group, in the case of the C' volunteer group, sequential distribution is more appropriate than parallel because the C' volunteer group frequently suffers from volunteer autonomy failures owing to a low α_v.

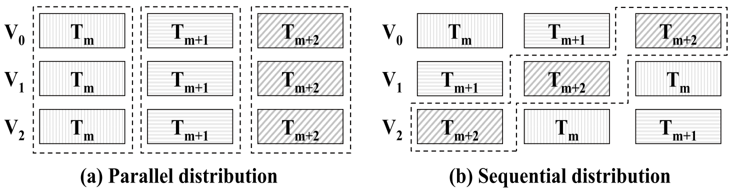


Fig. 3. Parallel and sequential distribution

3.4 Group Based Result Certification Mechanism

Result certification is dynamically applied to each volunteer group as follows: the A' volunteer group has high possibility that produce 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. For example, in the case of the T_{m+2} task in Fig. 3(b), if first two results generated at V_1 and V_2 are same, there is no need to execute the T_{m+2} task at V_0 because majority (i.e., 2 out of 3) is already achieved. As a result, other tasks can be executed instead of the executions that the solid line in Fig. 3(b) includes. The B' volunteer group has high possibility that produce correct results, but it cannot complete their tasks because of lack of the computation time. Moreover, volunteer autonomy failures occur frequently in the middle of execution. In the case of task migration, a previous volunteer affects the result of the volunteer to which a task is migrated. Accordingly, the migrated volunteer must be chosen among the B' or A' volunteer groups. The sequential voting group is appropriate like the case of the A' volunteer group. The C' volunteer group has enough time to execute tasks, but its results might be incorrect. To strength the credibility, the C' volunteer group requires more spot-checking and redundancy than the A' or B' volunteer group. The parallel voting group is more appropriate than the sequential voting group. Lastly, the D' volunteer group has insufficient time to execute tasks and there is little possibility to produce correct results. Moreover, volunteer autonomy failures occur frequently in the middle of execution. Accordingly, it is beneficial that tasks are not allocated to this volunteer group.

According to the above strategies, each S-MA has its own scheduling algorithm for result certification. In general, the tasks are scheduled in the following order: A', C', and B'.

The S-MA performs scheduling for result certification as follows: 1) Order each volunteer group by α_v, Θ, and C_v. 2) Evaluate the number of redundancy or spot-checking rate. 3) Construct a sequential voting group, or choose some volunteers for spot-checking on the basis of Θ and C_v . 4) Distribute tasks in a way of sequential voting group, or allocate special tasks for spot-checking. 5) Check the collected results.

In second phase, the number of redundancy for majority voting and the number of spot-checking are differently applied to each volunteer group. The number of redundancy for majority voting is dynamically regulated by each scheduling agent. The final error rate of majority voting [7] is evaluated by

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

which is bounded by $\frac{[4C'_v(1-C'_v)]^{k+1}}{2(2C'_v-1)\sqrt{\pi k}}$. Here, the parameter C'_v means the probability that volunteers within each volunteer group generate correct results.

Consider the desired credibility threshold ϑ. Our mechanism calculates the number of redundancy 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.

The rate of spot-checking q is also regulated by each scheduling agent. The final error rate of spot-checking [1] is evaluated by

$$\varepsilon(q, n, C'_v, s) = \frac{sC'_v(1 - qs)^n}{(1 - C'_v) + C'_v(1 - qs)^n} \tag{3}$$

where, n and s are the saboteur's share in the total work and the sabotage rate of a saboteur, respectively.

In a similar way of majority voting, if n and s are given, the spot-checking rate q of each volunteer group can be calculated using (3). Our mechanism calculates the rate of spot-checking for each volunteer group when $(1 - \vartheta) \geq \varepsilon(q, n, C'_v, s)$. The rate of spot-checking for the A' and B' volunteer groups are smaller than that of the C' volunteer group. Accordingly, the A' and B' volunteer groups can reduce the overhead, and thus execute more tasks. The C' volunteer group can increase its credibility.

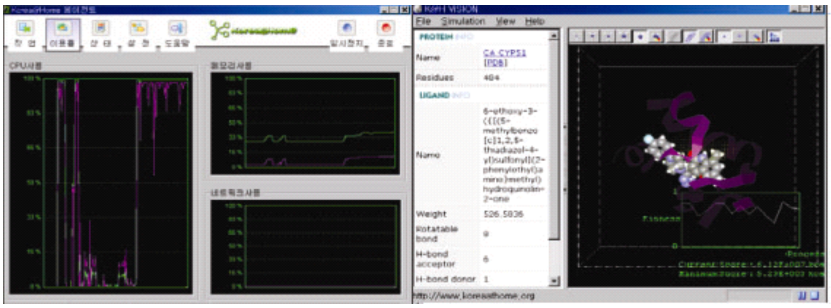
4 Implementation and Evaluation

4.1 Implementation Status

We have developed the "Korea@Home" [9], which attempts to harness the massive computing power of the great numbers of PCs distributed over Internet. Fig. 4 shows an execution screen shot in Korea@Home. Volunteers can take part in one of four kinds of applications: new drug candidate discovery, rainfall forecast, climate prediction, and optical analysis of TFT-LCD. The CPU types of volunteers are somewhat various, but the majority demonstrates similar CPU performance. For example, the Intel Pentium 4 consists of approximately 58% of the total, the Pentium III represents approximately 13%, the Celeron represents approximately 4%, and so on.

4.2 Simulations

We compare our group-based adaptive scheduling, result certification, and replication mechanisms with eager scheduling. For three kinds of cases, we evaluate 200 volunteers during one hour (see Table 1). Case 1 is different from Case 2 with



Name	CA CYP51 [PDB]
Residues	484
Name	6-ethoxy-3-(([(5-methylbenzo[c]1,2,5-thiadiazol-4-yl)sulfonyl](2-phenylethyl)amino)methyl)hydroquinolin-2-one
Weight	526.5836
Rotatable bond	9
H-bond acceptor	6
H-bond donor	1

Fig. 4. Screen shot of Korea@Home

Table 1. Simulation Environment

		A'	B'	C'	D'	Total
Case 1	# of vol.	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	# of vol.	71 (35.5%)	31 (15.5%)	76 (38%)	22 (11%)	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 3	# of vol.	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

# of vol.: the number of volunteers

regard to volunteer availability and volunteer availability. On the other hand, Case 3 is different form Case 1 with respect to volunteer availability and volunteering service time. Each simulation are repeated 10 times for each case. For simulation, the mean volunteering time of volunteers is selected in the range [10, 60] min. We also assume that $MTTVAF$=1/0.2~1/0.05 min. and $MTTR$=3~10 min. A task in the application exhibits 18 minutes of execution time on a dedicated Pentium 1.4GHz. The s and n for spot-checking are assumed to be 0.1 and 10, respectively.

Fig. 5 shows total number of completed tasks for scheduling mechanism with or without result certification. In this figure, ES and GAS represents eager scheduling and our mechanism, respectively. From Fig. 5, we observe that our mechanism completes more tasks than eager scheduling for all cases. In particular, the A' volunteer group has an important role in obtaining better performance. As the number of members in the A' volunteer group increases gradually (i.e., from Case 3 to Case 1), the number of completed tasks becomes higher. In contrast, as the number of members in D' volunteer group increases, the number of completed tasks becomes lower. Also, we see that volunteer availability

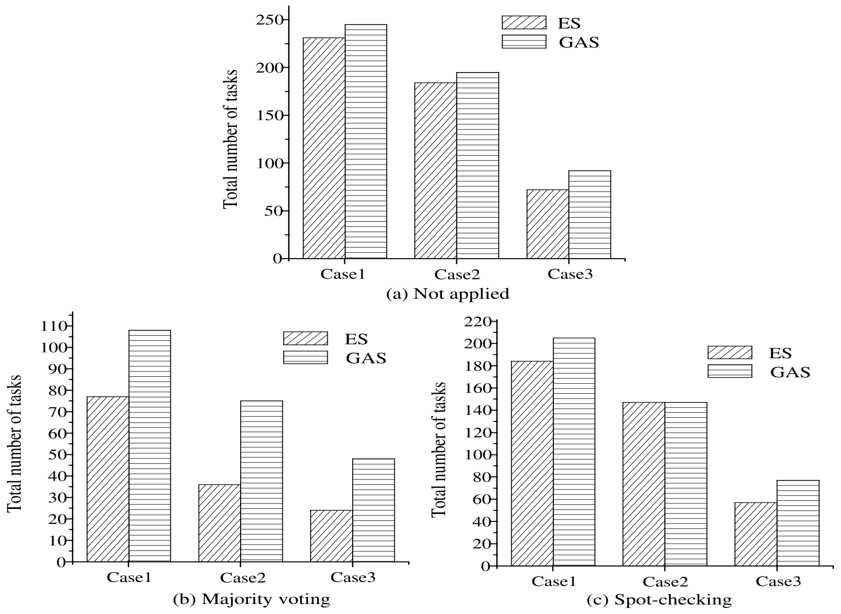


Fig. 5. The number of completed tasks

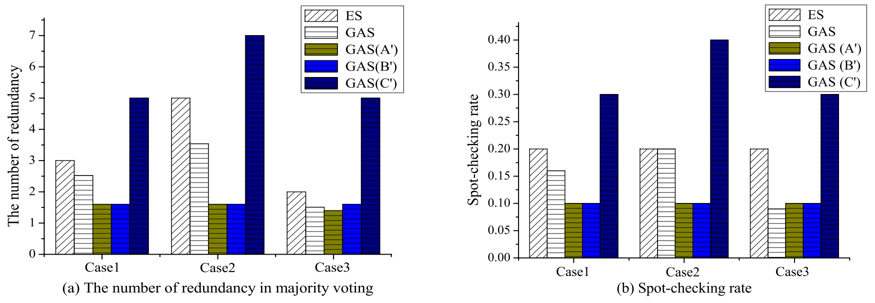


Fig. 6. The number of redundancy & spot-checking rate

is tightly related with performance; e.g., Case 1 can complete more tasks than Cases 2 and 3.

In the case of majority voting, our mechanism obtains more results of tasks than eager scheduling because it dynamically decides the number of redundancy according to properties of volunteer groups (see Fig. 6(a)). 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 the replication overhead, and so they can execute more tasks. The result of spot-checking is similar to that of majority voting (see Fig. 6(b)). This is because our mechanism can dynamically decide spot-checking rate according to properties of volunteer groups.

5 Conclusion

In this paper, we proposed a new peer to peer grid computing system based on mobile agents, which adapts to a dynamic environment. The proposed system applies different scheduling, result certification, and replication mechanisms to volunteer groups. As a result, it can reduce the overhead of a volunteer server by using adaptive mobile agents for each volunteer group in a distributed way. Moreover, the group based scheduling, replication, result certification mechanisms can complete more tasks than existing mechanism.

References

1. Sarmenta, L.F.G.: Sabotage-tolerance mechanisms for volunteer computing systems. Future Generation Computer Systems 18, 561–572 (2002)
2. Neary, M.O., Cappello, P.: Advanced eager scheduling for Java-based adaptive parallel computing. Concurrency and Computation: Practice and Experience 17, 797–819 (2005)
3. Kondo, D., Chien, A.A., Casanova, H.: Resource management for rapid application turnaround on enterprise desktop grids. In: ACM Conf. on High Performance Computing and Networking, pp. 19–30 (2004)
4. Lo, V., Zhou, D., Zappala, D., Liu, Y., Zhao, S.: Cluster computing on the fly: P2P scheduling of idle cycles in the Internet. In: Voelker, G.M., Shenker, S. (eds.) IPTPS 2004. LNCS, vol. 3279, pp. 227–236. Springer, Heidelberg (2005)
5. Lo, V., Zhou, D., Zappala, D., Liu, Y., Zhao, S.: Oddugi mobile agent system (2004), http://oddugi.korea.ac.kr
6. Ghanea-Hercock, R., Collis, J.C., Ndumu, D.T.: Co-operating mobile agents for distributed parallel processing. In: Proc. of the Third Int. Conf. on Autonomous Agents (AA 1999), pp. 398–399 (1999)
7. Zuev, Y.A.: On the estimation of efficiency of voting procedures. Theory of Probability & Its Applications 42, 78–81 (1998)
8. Li, Y., Mascagni, M.: Improving performance via computational replication on a large-scale computational grid. In: 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, pp. 442–448 (2003)
9. Li, Y., Mascagni, M.: Korea@home (2003), http://www.koreaathome.org/eng/
10. Trivedi, K.S.: Probability and Statistics with Reliability, Queuing and Computer Science Applications. Wiley, Chichester (2002)