Using Replication and Checkpointing for Reliable Task Management in Computational Grids

Sangho Yi, Derrick Kondo
INRIA, France
{*sangho.yi, derrick.kondo*}*@inria.fr*

Bongjae Kim, Geunyoung Park, Yookun Cho
Seoul National University, Korea
{*bjkim,gypark,cho*}*@os.snu.ac.kr*

Abstract

In grid computing systems, providing fault-tolerance is required for both scientific computation and file-sharing to increase their reliability. In previous works, several mechanisms were proposed for grid or distributed computing systems. However, some of them used only space redundancy (hardware replication), and others used only time redundancy (checkpointing and rollback). For this reason, the existing mechanisms are inefficient in terms of their resource utilization on grid systems. In this paper, we present *ART*, which is an Adaptive, Reliable, and fault-Tolerant task management for grid computing environments. The main goal of *ART* is reducing the number of replications by using checkpointing and rollback scheme for each replication. In *ART*, the minimum number of replications is adaptively selected based on analysis of probability of successful execution within the given deadline and reliability requirement of each task. Our simulation results show that *ART* can significantly reduce the number of replications and improve scalability compared with existing mechanisms.

1 Introduction

Rapid advancement in computer technologies enable us to use distributed grid computing environments for large scientific computation and resource-sharing services. The grid environments consist of many partipant nodes, and they have been used for weather forecasting file-sharing real-time multimedia broadcasting [1], and even nano-computing [2]. For example, *K*Grid* project [3] is an initiative in grid researches in Korea. The main goal of *K*Grid* is to provide an extremely powerful research environment to both academies and industries. In case of *TeraGrid*'s [4, 5] computing environment, it consists of over $100,000$ processors, and they compute large tasks collaboratively.

In above large-scale grid computing environments, failure occurs very frequently, because the failure rate is proportional to the number of processors (or, computing nodes). In [6], the authors studied that system failure occured 3 times per day on a large computing system which has $4,096$ processors. In addition, the *TeraGrid* system suffered from system failures every 2 minutes [7]. Even if the number of failures or the failure rate of each processor is very low, it may significantly affect the whole system's reliability. Without considering such failures, the grid system cannot be used for reliability-critical tasks. Therefore, large grid computing systems should have some fault-tolerance mechanisms to allow reliable execution of tasks.

Various research efforts have been made to provide fault-tolerance mechanisms for large grid computing environments. However, all of the existing schemes adapted only either over-provisioning or checkpointing and recovery mechanism. Some existing schemes [8, 9] used checkpointing and recovery, and they did not consider utilizing hardware replication. In [7, 10], they used both over-provisioning and task migration schemes to provide higher reliability, but they failed to minimize the degree of over-provisioning of each task. The two schemes seperately used in existing works, *over-provisioning* and *checkpointing* are *orthogonal*, because the over-provisioning requires more hardware and space, while the checkpointing requires more execution time. For this reason, it is necessary to use both the two orthogonal schemes appropriately for maximizing the reliability of large grid computing environments.

In this paper, we present *ART*, which is an Adaptive, Reliable, and fault-Tolerant task management for large grid computing environments. *ART* can reduce the number of replications (or, the degree of over-provisioning) by using a checkpointing and rollback scheme for each replication of a task. In *ART*, the minimum number of replications is adaptively selected based on probability analysis of successful execution within the given deadline and reliability requirement of each task. We made simulations for *ART* and existing fault-tolerance mechanisms, and the results show that *ART* significantly improves resource utilization and scalability when compared with the existing mechanisms.

The remainder of this paper is organized as follows. Section 2 shows previous work related to the fault-tolerance and reliability of the grid computing environments. Section 3 presents the design and internal structures, probability analysis, and algorithm of *ART* in detail. Section 4 evaluates performance of *ART* and the previous fault-tolerance schemes in terms of the resource utilization, scalability, and the number of replications. Finally, section 5 presents some conclusions with possible future work.

2 Related Work

In this section, we describe existing work related to task scheduling in grid computing environments. Several efforts have been made to provide reliability and fault-tolerance for the grid computing environments.

In [7], Kandaswamy et al. proposed *FTR*, which mainly uses over-provisioning to provide higher reliability with given parameters including success probability, execution time, failure probability of each computing node. They also considers using migration when the computation has failed, but they failed to reduce the degree of over-provisioning.

In [11], Elnozahy et al. summarized the recent rollback recovery protocols in message-passing systems. In this work, they showed the survey of distributed checkpointing scheme and recovery to the global consistent state of the distributed systems.

In [12], Duda proved the optimal checkpointing interval on the off-line when the checkpointing cost is constant. In case of using regular (or, full) checkpointing scheme, the checkpointing cost can be assumed as constant, and thus, we can calculate optimal checkpointing period based on this work. However, in incremental checkpointing, the checkpointing cost is varied under its amount of modified pages of a process, and the modification pattern of memory pages are not deterministic. To solve the problem, in [13], Yi et al. proposed an efficient and adaptive page-level incremental checkpointing facility that is based on the interval determination mechanism for minimizing the expected execution time. Using those, we can take full or incremental checkpoints with cost-efficient checkpointing interval.

In [14, 10], several fault-tolerance algorithms were tested on distributed, decentralized, and grid computing environments. Especially, in [14], they measured performance and reliability of the existing grid computing environments, *TeraGrid*. The results in [14] showed that the success ratio of execution in *TeraGrid* is about 58% for actual 6 weeks of operation. In [10], Alonso et al. presented the impact of fault-tolerance mechanisms on the real workflow management systems. In their reports, they showed that several kinds of replication-based fault-tolerance mechanisms can significantly increase the whole systems' reliability. Therefore, grid computing environments should have fault-tolerance mechanisms to increase their reliability.

3 ART: Adaptive, Reliable, and fault-Tolerant Task Scheduler

In this section, we present some requirements for grid computing environments and their applications. Then, we present system model, assumptions, and design of the *ART* in detail.

3.1 Requirements for Grid Computing Environments

Table 1 shows specification of the several well-knowns grid computing environments. In the most recent grid computing environments, more than thousands of computing nodes, or processors compose the whole system. Such grid environments may be tightly or loosely coupled according to their architectures or structures. In those grids, the most important resource is the computing power, or in other words, processors. Therefore, we need to consider *efficiency-in-task-scheduling* as the most important thing on the grid.

The second most important feature on the grid computing environments is *reliable task management*. On the grids, failure occurs frequently, and such failure may spoil performance and reliability. Research has shown [6, 7], that multiple failures may occur in a day when the number of processors is $4,096$, and failure occurs every 2 minutes when there are more than $100,000$ processors. In these cases, the successful execution ratio is significantly degraded by the frequent failures. Therefore, we need to provide *reliable task management* by using some appropriate reliability assurance mechanisms.

When we design a fault-tolerant task management for the grid environments, we also need to consider their *scalability*. This is closely related to the efficiency of the task management. If we use over-provisioning and/or checkpointing, some space or time overhead may be produced from the used fault-tolerance mechanisms. To provide high *scalability* with fault-tolerance, we need to minimize such space or time overhead. In other words, we need to determine the degree of over-provisioning as small as possible while minimizing the checkpointing and rollback overhead if we use checkpoints.

Table 1: Specification of well-known grid computing environments

Specification	*K*Grid*	*TeraGrid*	*LEAD*	*Pegasus*
Middleware	*MoreDream*	*Globus*	*ESML*	*DAGMan*
Structure	hierarchical	hierarchical	hierarchical	centralized
Number of computing nodes	7 supercomputers 9 clusters	about $116,000$	about $4,096$	$unknown$
Fault-tolerance support	checkpointing	migration & checkpointing	replication & migration	migration & retrying
Target application	bioinformatics molecular simulation	astronomy apps genome analysis	meteorology forecasting	genome analysis data mining

3.2 System Model and Assumptions

Some common notations and functions used throughout this paper are presented in Table 2.

The following is a system model used for our reliability analysis on the grid computing environments. Let t_r denote the work requirement time of a task, and the expected total execution time $T(t_r)$ of a task is defined as the processing time from the beginning of its execution to its completion. Note that in the absence of any failures without checkpointing, $T(t_r) = t_r$.

The $T(t_r)$ can be calculated as the sum of all intervals' expected execution time, $T_i(t_i, c_i)$, where the t_i and c_i are the work requirement time and checkpointing overhead of an i^{th} interval. Note that, a checkpointing interval is the duration between two checkpoints. Each interval begins when a checkpoint is established and ends when the next checkpoint is established.

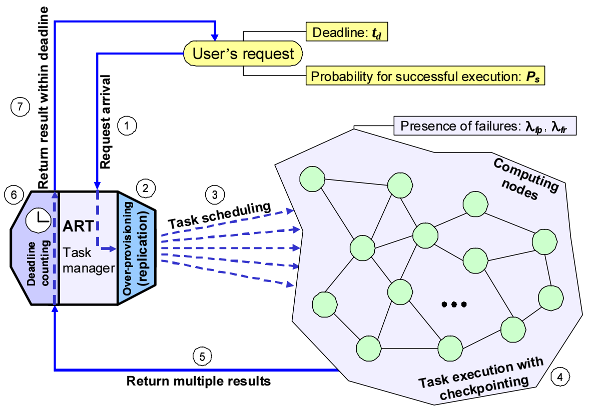
t_d represents the deadline for a task, and P_s is a user-defined probability of successful execution of a task within the deadline. $P(t_r, t_d)$ is probability of successful execution of a task which has t_r as the work requirement time and t_d as the deadline.

In this paper, we used the following assumptions. We assume that two kinds of failures can occur on the grid computing environment: including permanent and recoverable failures. The permanent failure is similar to hardware failures, and thus, we cannot recover the consistent state based on checkpointing and rollback mechanism if the permanent failure occurs on a system. On the other hand, the recoverable failures can be recovered by using the last checkpoint. We assume that the permanent and recoverable failures occur according to *Poisson* process at rate λ_{fp} and λ_{fr}, respectively. These are commonly accepted assumptions, particularly when failures are known to occur as a result of many different reasons [13]. Further, we assume that recoverable failures are detected as soon as they occur.

3.3 Design of ART

ART provides fault-tolerant task processing based on both the space and time redundancies[1]. The space redundancy means the over-provisioning (or, hardware replication) technique, while the time redundancy means the checkpointing and rollback scheme. In *ART*, the minimum set of replications is determined based on the probability of successful execution with checkpointing and given deadline of each task.

3.3.1 Overview of ART



User's request
Deadline: t_d
Probability for successful execution: P_s
7
Return result within deadline
1
Request arrival
Presence of failures: λ_{fp}, λ_{fr}
6
2
3
Task scheduling
Computing nodes
Deadline counting
ART
Task manager
Over-provisioning (replication)
5
Return multiple results
Task execution with checkpointing
4

Figure 1: Overall structures of *ART*

Figure 1 shows the overall structures and the example flow of the proposed mechanism. Each computing node uses checkpointing and rollback scheme to provide fault-tolerance in the presence of failures. The task management in *ART* uses over-provisioning to increase reliability of executing task requests. When a user's request occurs, the request has the predefined values of deadline and required probability (or, reliability) for successful execution. Then, *ART*'s task management routine se-

[1]The preliminary poster paper was presented in [15].

Table 2: List of notations used in this paper

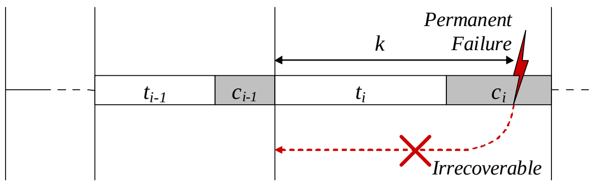
Notation	Description
t_r	total work requirement time of a task
t_d	deadline time of a task
t_l	laxity time of a task
t_i	work requirement time of interval i
c_i	checkpointing cost of interval i
n_c	number of checkpoints of a task
r	recovery cost of a task
λ_{fp}	permanent failure rate
λ_{fr}	recoverable failure rate
$T_{r_i}(t_i, c_i)$	expected recovery and reprocessing time of interval i of a task
$T_r(t_r)$	total expected recovery and reprocessing time of a task
$\phi_{fr}(t_r, t_d)$	probabilistic density function of successful execution of a task requirement t_r within deadline t_d in the presence of recoverable failures
$\Phi_{fr}(t_r, t_d)$	cumulative density function of successful execution of a task requirement t_r within deadline t_d in the presence of recoverable failures
$\Phi_{fp}(t_r)$	cumulative density function of successful execution of a task requirement t_r in the presence of permanent failures
P_s	user-defined probability of successful execution of a task
$P(t_r, t_d)$	probability of successful execution of a task within deadline

lects computing nodes to schedule the task request while meeting the two constraints in terms of the deadline and the reliability.

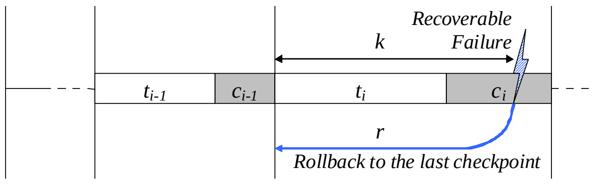
To decide how many replications are used in the task execution, we need to calculate the expected execution time and the probability for successful execution of the task for each computing node within the given deadline. We present the above calculation and probability analysis in the following subsections.

3.3.2 Expected Recovery and Reprocessing Time in ART

Figure 2 shows two examples of failures when executing a task on a computing node. In case of Fig. 2 (a), a permanent failure occurs, and then the computing node cannot be recovered by the checkpointing and rollback mechanism. Figure 2 (b) shows an example of recoverable failure occurance on the computing node. When a recoverable failure occurs during an i^{th} checkpointing interval, the process can perform a rollback operation and reprocess its execution from the beginning of the i^{th} interval. If no failure occurs, the execution time of the i^{th} interval is identical to $(t_i + c_i)$. However, if a failure occurs before the end of the i^{th} interval, the recovery time r is required. The expected execution time of the i^{th} interval with checkpointing, $T_{r_i}(t_i, c_i)$ is given as Theorem 1 [13].



(a) An example of a permanent failure

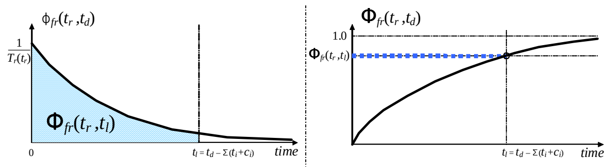


(b) An example of a recoverable failure

Figure 2: Two examples of failure occurance on a computing node

Theorem 1 *If we assume the recoverable failures occur according to the Poisson process with rate λ_{fr} and that failures can occur during checkpointing, the expected execution time of the i^{th} interval with checkpointing, $T_{r_i}(t_i, c_i)$ is given by*

$$T_{r_i}(t_i, c_i) = \frac{(e^{\lambda_{fr}(t_i+c_i)} - 1)(1 + \lambda_{fr} r) - \lambda_{fr}(t_i + c_i)}{\lambda_{fr}} \quad (1)$$



(a) Probabilistic density function (b) Cumulative density function

Figure 3: An example of the probabilistic density function (pdf) and cumulative density function (cdf) of the recovery and reprocessing time of a task requirement t_r (where t_d is the given deadline and t_l is the calculated laxity time of the task requirement)

Proof 1 *Formally, the conditional expected recovery and reprocessing time is written as:*

$$T_{r_i}(t_i, c_i) = \begin{cases} 0 & \textit{if} \quad k \geq t_i + c_i \\ k + r + T_{r_i}(t_i, c_i) & \textit{otherwise} \end{cases}$$

By the law of total expectation,

$$T_{r_i}(t_i, c_i) = \int_0^{t_i+c_i} (k + r + T_{r_i}(t_i, c_i)) f_r(k) dk$$

Solving the above equation we obtain,

$$T_{r_i}(t_i, c_i) = \frac{\int_0^{t_i+c_i} (k+r) f_r(k) dk}{1 - \int_0^{t_i+c_i} f_r(k) dk}$$

Finally the probabilistic density function of the recoverable failure $f_r(k)$ is $\lambda e^{-\lambda_{fr} k}$, hence we obtain,

$$T_{r_i}(t_i, c_i) = \frac{(e^{\lambda_{fr}(t_i+c_i)} - 1)(1 + \lambda_{fr} r) - \lambda_{fr}(t_i + c_i)}{\lambda_{fr}}$$

When the number of checkpointing intervals is n_c for the task requirement t_r, the expected total execution time with checkpointing can be expressed as follows.

$$T_r(t_r) = \sum_{i=1}^{n_c} T_{r_i}(t_i + c_i) \tag{2}$$

In Eq.(1) and Eq.(2), when the checkpoints are equally spaced and the checkpointing cost is constant c, the equations become much simpler. In case of using the full checkpointing, the assumption of the constant checkpointing cost c can be almostly accepted. However, these assumptions do not fully reflect the real characteristics of incremental checkpointing. In that case, we can determine the efficient checkpointing intervals based on the previous adaptive interval determination scheme [13].

3.3.3 Probability Analysis for Successful Execution in ART

Using the above equations, we analyze the probability for successful execution when using the checkpointing mechanism. Since we assumed that failures occur according to a *Poisson* process, the probabilistic density function of the expected recovery and reprocessing time follows the *Poisson* distribution [2].

In Fig. 3 (a), the sum of colored (or, dashed) region represents the probability for successful execution within the deadline in the presence of recoverable failures. Figure 3 (b), shows the cumulative probability for successful execution of the task requirement t_r within the deadline t_d. The probability for successful execution within the deadline in the presence of both permanent and recoverable failures is given as Theorem 2.

[2]We are currently focused on deriving proof of the *Poisson* distribution for probabilistic density function of the recovery and reprocessing time.

Theorem 2 *If we assume both permanent and recoverable failures occur according to the Poisson process with rate λ_{fp} and λ_{fr}, respectively, the probability for successful execution for the time requirement t_r within the deadline t_d, $P(t_r, t_d)$ is given by*

$$P(t_r, t_d) = e^{-\lambda_{fp} t_d} \cdot \left(1 - e^{-\frac{t_d - \sum_{i=1}^{n_c}(t_i+c_i)}{T_r(t_r)}}\right) \tag{3}$$

Proof 2 *According to the* Poisson *distribution, the probability density function (pdf) of the recovery and reprocessing time of a task requirement t_r within the deadline t_d, $\phi_{fr}(t_r, t_d)$ is written as:*

$$\phi_{fr}(t_r, t_d) = \frac{1}{T_r(t_r)} \cdot e^{-\frac{t_d - \sum_{i=1}^{n_c}(t_i+c_i)}{T_r(t_r)}}$$

By integrating the above equation we obtain,

$$\Phi_{fr}(t_r, t_d) = 1 - e^{-\frac{t_d - \sum_{i=1}^{n_c}(t_i+c_i)}{T_r(t_r)}}$$

Also, the permanent failures occur according to the Poisson *process at rate λ_{fp}, the probabilistic density function of the permanent failure at time t_d becomes $\lambda_{fp} e^{-\lambda_{fp} t_d}$. By integrating this, we obtain the cumulative density function of a permanent failure occurance at time t_d as follows:*

$$F_{fp}(t_d) = 1 - e^{-\lambda_{fp} t_d}$$

Then, $1 - F_{fp}(t_d)$ becomes the probability for successful execution during the time t_d when the permanent failures can occur on a computing node at rate λ_{fp}. By multiplying the $\Phi_{fr}(t_r, t_d)$ and the $1 - F_{fp}(t_d)$, we obtain,

$$P(t_r, t_d) = e^{-\lambda_{fp} t_d} \cdot \left(1 - e^{-\frac{t_d - \sum_{i=1}^{n_c}(t_i+c_i)}{T_r(t_r)}}\right)$$

3.3.4 Selection of Replications in ART

The above $P(t_r, t_d)$ represents the reliability of each computing node when processing a task requirement t_r within a deadline t_d. In general, grid computing environments consist of heterogeneous computing nodes, and each node has different computing power. In addition, the failure rates, λ_{fp} and λ_{fr} can be different each other. For this reason, each node has a different value of the $P(t_r, t_d)$. In this subsection, we present how to select computing nodes to replicate the requested task.

Selecting an optimal set of computing nodes is a **NP-hard** problem. For this reason, the existing scheme [7] used a heuristic approach to get sub-optimal set of nodes to replicate tasks. In this paper, we also use similar heuristics to compare our method with it. We are currently extending our work to design better algorithm to minimize resource consumption on the grid computing environments.

Algorithm 1 Adaptive replication selection algorithm

```
Require:
  (where $n_i$ is an $i^{th}$ computing node, and $m$ is the total number of nodes)
  for all nodes $n_i$ (from 1 to $m$) do
    Calculate $n_i.P(t_r, t_d)$ using Theorem 2
  end for
  Sort all nodes by $n_i.P(t_r, t_d)$ in descending order
  $P_{tmp} := 1$
  for all nodes $n_i$ (from 1 to $m$) do
    $P_{tmp} := P_{tmp} \cdot (1 - n_i.P(t_r, t_d))$
    if $P_s < 1 - P_{tmp}$ then
      $k := i$
      break
    end if
  end for
  for nodes $n_i$ (from 1 to $k$) do
    Do task replication
  end for
```

Algorithm 1 describes the adaptive replication selection algorithm in detail. It consists of two parts; the first part calculates the $P(t_r, t_d)$ for all computing nodes, and the second pard is used to select some node to replicate the requested task. Using this algorithm, we determine the minimum set of replications at run-time.

3.4 Remaining Issues in ART

We assumed that failure occurs according to the *Poisson* process. If the failures occur with non-exponential distribution, (e.g. Gamma, Weibull, etc.), the equations on Theorems 1 and 2 should be changed to calculate $P(t_r, t_d)$). If it is very hard to use the existing probabilistic distribution, we can get the *nearest model* or the combination of existing models based on the real failure traces, which was already shown in [16, 17].

In addition, non-heuristic selection algorithm based on the statistics or history may improve both the reliability and efficiency of the grid computing environments.

4 Performance Evaluation

In this section, we evaluate the performance of *ART* and existing fault-tolerant mechanisms in terms of resource utilization, successful execution ratio, and scalability. We made simulations to evaluate the several mechanisms from various point of view.

4.1 Simulation Environment

In this paper, we modified and used the *SensorMaker* [18] simulation package, and we set the default values of defined parameters as presened in Table 3.

In our simulation, each task request has different time requirement and deadline. We used the fixed checkpointing interval, fixed costs for checkpointing and recovery, and $1-1.0\cdot10^{-12}$ as the probability for successful execution. In addition, each computing node has different failure rates, and the λ_{fr} and the λ_{fp} were determined based on previous researches [6, 7]. In Table 3, m is the total number of computing nodes, and k is the task scheduling request arrival interval. Finally, we assumed that each computing node processes tasks in a *FIFO* manner (one task at a time).

4.2 Simulation Results and Evaluation

Based on the default values presented in Table 3, we made various simulations in many aspects by changing values of several parameters.

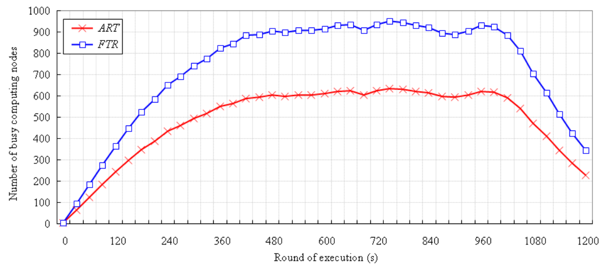


Figure 4: Number of busy (running) computing nodes

Figure 4 shows the number of busy (or, running) computing nodes when using *ART* and *FTR*, respectively. In this result, *ART* outperforms the previous scheme for all time, and reduces the number of used nodes significantly.

Table 3: Values of parameters used in performance evaluation

Parameter	Value	Description
t_r	$100 \sim 500s$	each task has one of between 100 and $500s$ time requirement
t_d	$200 \sim 1000s$	two times of t_r
t_i	$50s$	fixed checkpointing interval
c_i	$3s$	fixed checkpointing cost
r	$2s$	fixed recovery cost
P_s	$1 - 1.0 \cdot 10^{-12}$	99.999...% probability for successful execution
λ_{fr}	$2.26 \sim 7.18 \cdot 10^{-8}$	variable recoverable failure rate per second [6, 7]
λ_{fp}	$2.26 \sim 7.18 \cdot 10^{-11}$	variable permanent failure rate per second
m	$2,000$	total number of computing nodes
k	$1s$	task scheduling request arrival interval

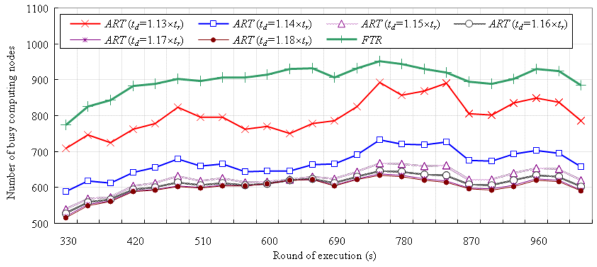


Figure 5: Number of busy computing nodes according to the changed deadline

Figure 5 represents the number of busy nodes according to the deadline, t_d. This result shows that, the longer the value of the deadline, the greater the reduction in computing node usage. For example, when the deadline is 1.18 times greater than the time requirement, *ART* uses only $\frac{2}{3}$ of computing nodes as compared with the other scheme.

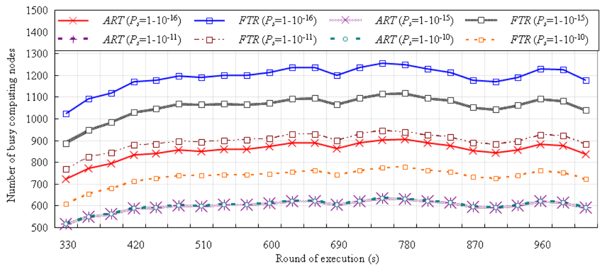


Figure 6: Number of busy computing nodes according to the changed probability for successful execution

Figure 6 shows the number of busy nodes according to the changed probability for successful execution, P_s. The P_s is positively proportional to the number of busy computing nodes. For example, higher P_s means higher reliability, and then, the task scheduler need to use more replication to fulfill the higher reliability. On the other hand, when P_s becomes smaller, then smaller number of computing nodes are used for replication.

Finally, Table 4 shows the maximum, minimum and average number of busy nodes during $300 \sim 1,200$ rounds. In this table, efficiency of *ART* is significantly better than the previous scheme. From the simulation results, we observe that the proposed scheme, *ART* outperforms previous schemes in terms of both efficiency and scalability.

5 Conclusions

In this paper, we proposed *ART* (Adaptive, Reliable, and fault-Tolerant task scheduler) for grid environments. *ART* reduced the number of replications by using checkpointing and rollback scheme for each replication. In *ART*, adaptive selection of the minimum number of replications is performed by analyzing probability of successful execution within the given deadline and reliability requirement of each task. We made simulations for *ART* and existing fault-tolerance mechanisms. Based on the results, we observe that *ART* significantly reduces the number of replications and improves scalability compared with existing mechanisms.

We are currently extending our work to design better selection algorithm. In addition, we are designing an appropriate checkpointing method which is required for maximizing efficiency of the grid computing environments in the future work.

6 Acknowledgments

This research was supported by the ANR project Clouds@home (ANR-09-JCJC-0056-01), and the Basic Science Research Program through the National Re-

Table 4: Maximum, minimum, and average number of busy computing nodes during $300 \sim 1,200$ rounds

Number of busy nodes	*ART*			*FTR*		
P_s	$1-10^{-16}$	$1-10^{-15}$	$1-10^{-11}$	$1-10^{-16}$	$1-10^{-15}$	$1-10^{-11}$
Maximum	904	634	632	$1,255$	$1,115$	943
Minimum	340	228	224	456	425	342
Average	790	554	549	$1,100$	977	827

search Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (KRF-2008-314-D00335).

References

[1] PeerLive, *http://www.peerlive.org*, website.

[2] C.-H. Huang, "High-performance parallel biocomputing," *Parallel Computing*, vol. 30, pp. 1037–1055, September-October 2004.

[3] K*Grid, *http://www.gridcenter.or.kr/*, website.

[4] C. Catlett, "The philosophy of teragrid: Building an open, extensible, distributed terascale facility," in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, 2002, p. 8.

[5] D. A. Reed, "Grids, the teragrid, and beyond," *Computer*, vol. 36, pp. 62–68, January 2003.

[6] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance-computing systems," in *International Conference on Dependable Systems and Networks (DSN'06)*, June 2006.

[7] G. Kandaswamy, A. Mandal, and D. A. Reed, "Fault tolerance and recovery of scientific workflows on computational grids," in *8th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'08)*, 2008, pp. 777–782.

[8] J. Weissman, "Fault tolerant computing on the grid: what are my options?" in *8th International Symposium on High Performance Distributed Computing (HPDC'99)*, August 1999, pp. 351–352.

[9] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, pp. 237–246, July 2002.

[10] G. Alonso, C. Hagen, D. Agrawal, A. E. Abbadi, and C. Mohan, "Enhancing the fault tolerance of workflow management systems," *IEEE Concurrency*, vol. 8, pp. 74–81, July 2000.

[11] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, pp. 375–408, September 2002.

[12] A. Duda, "The effects of checkpointing on program execution time," *Information Processing Letters*, vol. 16, no. 1, pp. 221–229, Jul. 1983.

[13] S. Yi, J. Heo, Y. Cho, and J. Hong, "Taking point decision mechanism for page-level incremental checkpointing based on cost analysis of process execution time," *Journal of Information Science and Engineering*, vol. 23, no. 5, pp. 1325–1337, September 2007.

[14] O. Khalili, J. He, C. Olschanowsky, A. Snavely, and H. Casanova, "Measuring the performance and reliability of production computational grids," in *7th IEEE/ACM International Conference on Grid Computing*, September 2006.

[15] S. Yi, H. Min, B. Kim, J. Kim, and C. O. Sung, "Art: Adaptive, reliable, and fault-tolerant task management for computational grids," in *2010 ACM Symposium on Applied Computing (ACM SAC'10)*, 2010, pp. 238–239.

[16] B. Javadi, D. Kondo, J. Vincent, and D. Anderson, "Mining for statistical availability models in large-scale distributed systems: An empirical study of seti@home," in *17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, September 2009.

[17] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," in *10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2010.

[18] S. Yi, S. Lee, Y. Cho, and J. Hong, "SensorMaker: A wireless sensor network simulator for scalable and fine-grained instrumentation," *Lecture Notes in Computer Science*, vol. 5072, pp. 800–810, 2008.