Improving Performance via Computational Replication on a Large-Scale Computational Grid

Yaohang Li and Michael Mascagni

Department of Computer Science and School of Computational Science and Information Technology
Florida State University
Tallahassee, FL 32306-4530, USA
{yaohanli, mascagni}@cs.fsu.edu

Abstract

High performance computing on a large-scale computational grid is complicated by the heterogeneous computational capabilities of each node, node unavailability, and unreliable network connectivity. Replicating computation on multiple nodes can significantly improve performance by reducing task completion time on a grid's dynamic environment. We develop an analytical model to determine the number of task replicas to meet the performance goals in different computational grid configurations. Furthermore, taking advantage of the statistical nature of grid-based Monte Carlo applications, we extend the computational replication technique to an N-out-of-M scheduling strategy for grid-based Monte Carlo applications, which can potentially form a large category of grid-computing applications. In addition, we establish a corresponding model for the N-out-of-M scheduling mechanism. Simulations are used to validate the computational replication models. Our preliminary results show that the models we use are effective in predicting the required number of replicas to achieve short task completion time with a given high probability.

1. Introduction

Grid computing is characterized by large-scale sharing and cooperation of dynamically distributed resources, such as CPU cycles, communication bandwidth, and data, to constitute a computational environment [1]. A large-scale computational grid can, in principle, offer a tremendous amount of low-cost computational power. This attracts many computationally intensive scientific applications. On the other hand, significant challenges also arise. Within a computational grid's dynamic environment, the computational capabilities of each node vary greatly. As a result, a task running on different nodes on the grid will have a huge range of completion times. Also, due to unreliable network connections and the possible unavailability of a node, an executing task may be delayed or even halted at any time. Therefore, from the grid-application point of view, how quickly a computational grid can complete a group of submitted tasks from an application becomes an issue of prime importance.

In this paper, we investigate a computational replication technique to develop an optimal scheduling mechanism to improve the throughput of a computational grid and reduce task completion time. This is different from the task-scheduling problem that has been discussed for many conventional parallel or distributed computing environments [2, 3], where there are a very limited number of nodes. In contrast, on a computational grid, the available computational service providers can essentially be treated as unlimited compared to the number of existing tasks. Therefore, we have more freedom to use these massive computational resources as trade-offs to achieve better task completion times.

The remainder of this paper is organized as follows. In Section 2, we provide an approach to apply the computational replication technique to a computational grid. We establish the analytical model of our replicate scheduling mechanism and evaluate our model using simulation in Section 3 and Section 4, respectively. Using simulation, we also compare the performance of replicate scheduling with that of dynamic rescheduling under different grid configurations. Section 5 extends the computational replication technique to the *N-out-of-M* scheduling strategy for Monte Carlo applications. Finally, Section 6 summarizes our conclusions and future research directions.

2. Computational Replication

Replication is a well-known technique for improving availability in an unreliable system. In fault-tolerant computing, replication is also a technique to overcome faults [4, 5]. The replication technique has already been favorably utilized in grid computing. In SETI@home [6], a majority voting mechanism[1] is applied to check the correctness of a task. Ranganathan, Iamnitchi, and Foster discussed using dynamic model-driven replication to obtain high data availability in a large peer-to-peer community [7]. In this paper, we are interested in improving the performance of a computational grid by replicate scheduling of grid tasks.

The basic idea of replicate scheduling in a computational grid is concurrently executing multiple copies of a given task. If multiple copies of a computational task are executed on independent nodes, then the chance that at least one copy is completed during a specific period of time increases. As a result, the time between submitting a task and obtaining a result is very probably reduced. Concurrent assignment of tasks to multiple nodes guarantees that a particular, very slow, machine will not slow the aggregate progress of a computation. Eventually, under the assumption of unlimited computational service providers available in the pool, the throughput of the computational grid will tend to increase with increasing numbers of computing replicas for each task.

The implementation of computational replication on a computational grid is rather simple. Figure 1 shows the mechanism of replicate scheduling in a grid-computing environment. When the computational grid receives a task, r copies are replicated and scheduled to r different nodes. At that point in time, r copies of the task are concurrently running. Once an execution is complete and the corresponding result is obtained, the task is regarded as finished. Termination signals can be sent to the other nodes to abort their current running jobs.

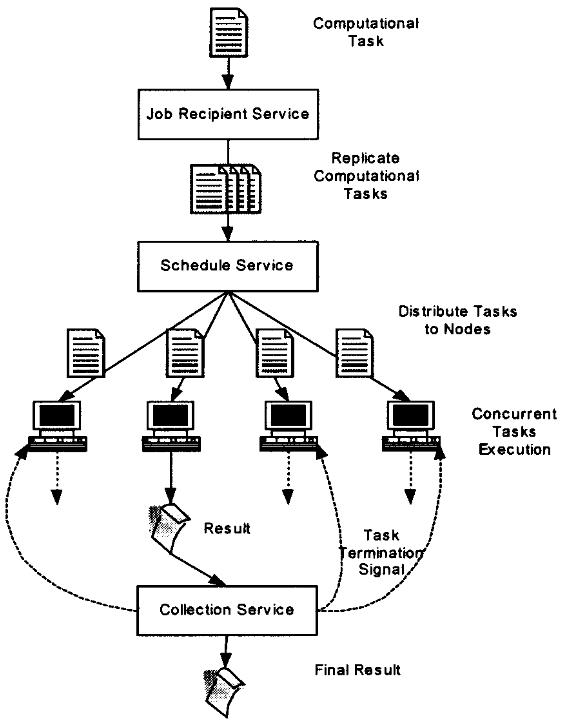

Figure 1: Replicate Scheduling in a Computational Grid

Using the computational replication technique can prevent slow or unstable nodes from slowing down or halting a grid task with high probability, which could lead to a reduced completion time of this task. However, we do not wish to imply that the more replicas, the better. The execution of too many copies of a task may not contribute much to reducing completion time but may significantly increase the grid system's workload. Such problems can be found in metacomputing prototype Charlotte [8] using its eager scheduling mechanism. Eager scheduling aggressively assigns and reassigns existing tasks to available nodes in the distributed-computing system to keep all the nodes busy. Nevertheless, the following phenomenon may occur: there may be many copies of a task running on the system and occupying many computational resources. However, the later arriving tasks may not be able to find an available node, which will reduce the system throughput. In short, to determine what is a "reasonable" number of replicas becomes critical for the computational replication technique. We will establish a system model to probe for answers to this question in the next section.

[1] Different nodes process the same copy of tasks independently and the final result is obtained by a majority vote of the distributed results

IEEE COMPUTER SOCIETY

3. Analytical Methods

To determine the number of computing replicas to achieve a specific performance requirement, we need to consider some system parameters. In a computational grid, the completion time of a grid task depends on the performance of each individual node participating in the computation, the node failure rate, and also the network failure rate. We make the following assumptions to set up our model.

1) The execution of a task completely occupies a node on the grid, and no other jobs can be executed on the same node concurrently.
2) Compared to the execution time (usually from hours up to days), the tasks' scheduling time and result collection time (usually in the range of seconds or minutes) is short enough to be ignored.
3) Each node works on its task independently.
4) Each node has an equal probability of obtaining a task from the schedule service. The tasks are scheduled without noticing the performance of each node.
5) A task is architecture-independent.

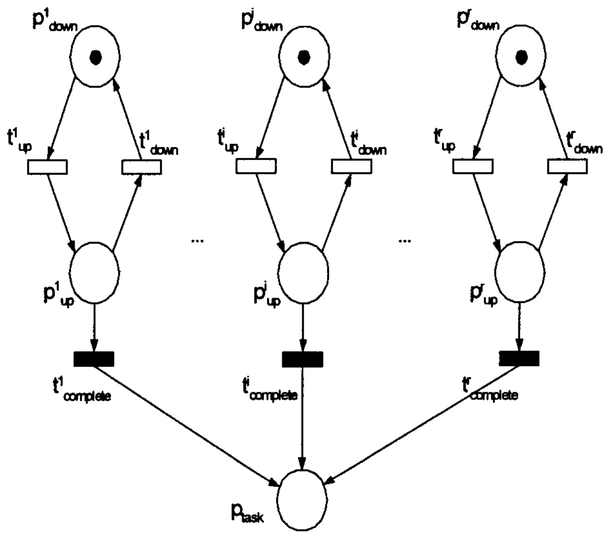

Figure 2: Petri Net Modeling of Computation Replication in the Grid

Figure 2 shows the Petri Net (PN) model of replicated tasks concurrently running on a computational grid. In this PN model, a node, i, alternates between an up state (place p^i_{up}) and a down state (place p^i_{down}). Transition t^i_{down} represents node unavailability (with unavailability rate λ) and transition t^i_{up} node back to service (with availability rate μ). Transition $t^i_{complete}$ is assigned the task progress threshold W (usually 100%) so that the task completion condition (token in p_{task}) is reached when W is hit.

Let r be the total number of computing replicas,

p^i_{sys} be the probability of node i participating in the computations is up, where $p^i_{sys} = \mu/(\mu+\lambda)$,

θ_i' be the service rate of node i, which can be measured as the number of tasks that can be finished within a specific period of time without interruption. Considering the node availability, the service rate, θ_i, in node i is $\theta_i = \theta_i' * p^i_{sys}$.

Then, the service time distribution function $S_i(t)$, referring to the probability that the task completion time T_i is less than t, which conforms to an exponential distribution, can be represented as

$$S_i(t) = \Pr(T_i \geq t) = \int_t^{\infty} \theta_i e^{-\theta_i x} dx = e^{-\theta_i t}.$$

The probability that a task can be completed by time t, which is the cumulative distribution function of the exponential distribution, is

$$p_{task_i}(t) = 1 - S_i(t).$$

Finally, the probability, $p_{replica}(t)$, that at least one will be done by time t is

$$p_{replica}(t) = 1 - \Pr(Min(T_1, T_2, \ldots, T_r) \geq t) = 1 - e^{-\sum_{i=1}^{r} \theta_i t}.$$

By evaluating the mean of the service rates θ_i, $\theta = \frac{1}{r}\sum_{i=1}^{r} \theta_i$, in each node participating in the computation, we are able to estimate a proper number of replicas, r. Suppose we want at least one task completion at time t with probability α, then, we need to have at least r copies of tasks running, where

$$r = \left\lceil \frac{\ln(1-\alpha)}{\theta t} \right\rceil.$$

4. Simulation Results

In our simulation program, we simulated a 1,000-node computational grid. Nodes join and leave the system with a specified probability. Also, nodes have a variety of computational capabilities. Each simulation is run for 1,000 time steps. (A task running on a node with service rate θ will take $1/\theta$ time steps, e.g., a fast node with service rate 0.01 will take 100 time steps to complete the task while a slow one with service rate 0.001 will take 1,000) At each time step, a certain number of nodes go

IEEE COMPUTER SOCIETY

down while a certain number of nodes become available for computation. We built our simulations in order to

1) evaluate the validity of our model, and to
2) compare the computational replication technique with the dynamic rescheduling technique.

4.1 Model Validation

Our model computes the minimum number of replicas that are necessary to achieve a certain task completion probability at a specified time. At the same time, our model can also evaluate the task completion probability using the computational replication technique. In order to validate the accuracy of our model, we therefore fix the number of replicas and compare the actual task completion rate with the predicted probability of our model at different time steps.

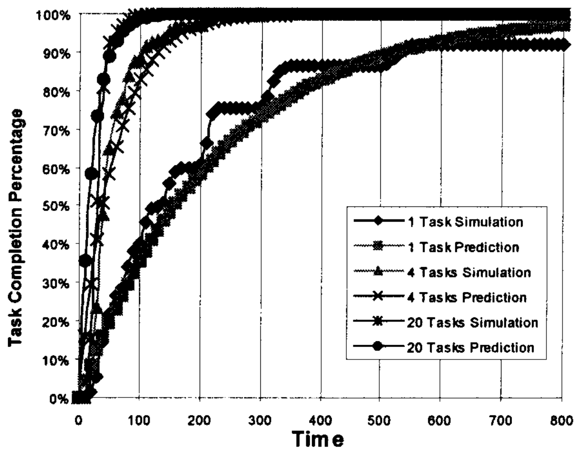

Figure 3: Model Prediction versus Actual Behavior with Different Numbers of Replicas

Figure 3 shows the comparison between the simulation results and the prediction from our analytical model. From the graph, we can see that the actual behavior matches our model prediction quite well. Also, we notice that with 1 task running, at least 600 time steps are required to obtain 90% task completion percentage, however, with 4 replicas, less than 200 time steps are required to obtain the same percentage. This indicates a significant task completion time reduction using the computational replication technique. However, we also found that even when we increase the number of current tasks to 20, we cannot significantly increase task performance. Therefore, with a proper number of replicas, we can achieve an optimal performance/cost ratio.

4.2 Replicate Scheduling vs. Dynamic Rescheduling

To prevent a slow node from delaying or halting the completion of a grid task, the dynamic rescheduling technique is another popular method used in existing computational grid systems like Condor [9, 13] and Entropia [10]. In dynamic rescheduling, the system keeps track of the execution of each task. When a task is halted, a checkpoint of the execution is then generated. Next, the schedule service will look for another available and appropriate node to reschedule the task. After the checkpoint data file is transferred to the new node, the task then continues to execute on the new node by recovering the execution process of the task. The dynamic rescheduling technique can keep the task running all the time but at the cost of additional system administration and rescheduling overhead involving task status monitoring, checkpointing, searching for available nodes, network transferring, task rescheduling, and execution recovering.

We simulate the scheduling mechanism using the dynamic rescheduling technique. When a node running a task is down, the task is rescheduled to another available node. The rescheduling penalty is taken into consideration when task rescheduling occurs in the simulation. Figures 4 and 5 illustrate the task completion time comparison between replicate scheduling and dynamic rescheduling. The data in Figure 4 come from simulation on a computational grid comprised of nodes with similar performance characteristics. This can be a grid constructed from computers in a computer lab that have similar performance parameters and are connected by a high-speed network. The cost of task rescheduling is relatively low in such a situation. Our simulation results show that the dynamic rescheduling technique has a better task completion time than that of replicate scheduling when the node unavailability rate is high. When the node down rate is low, both techniques have a similar simulated performance. Figure 5 simulates a computational grid whose nodes have computational capabilities in a wide range. In practice, this grid can be a system with geographically widely distributed nodes like SETI@home [6]. In this grid system, a node might be a high-end supercomputer, or a low-end personal computer, or even just an intelligent widget. The connection among nodes is via a low-speed network, which carries a high task rescheduling cost. We notice that in our simulation results, replicate scheduling using an appropriate number of replicas has a better task completion time than that of the dynamic rescheduling technique.

IEEE COMPUTER SOCIETY

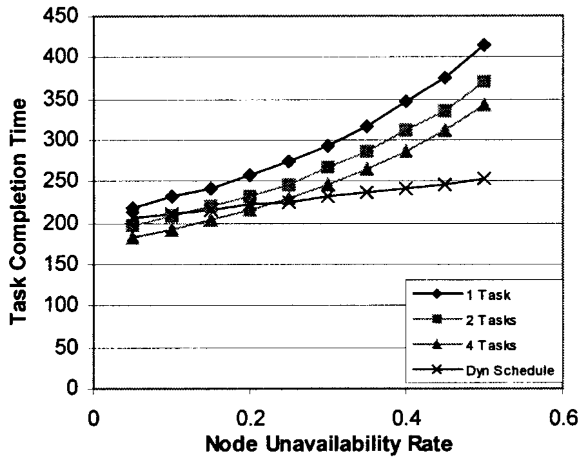

Figure 4: Replicate Scheduling vs. Dynamic Rescheduling on a Computational Grid with Nodes Sharing Similar Performance Parameters

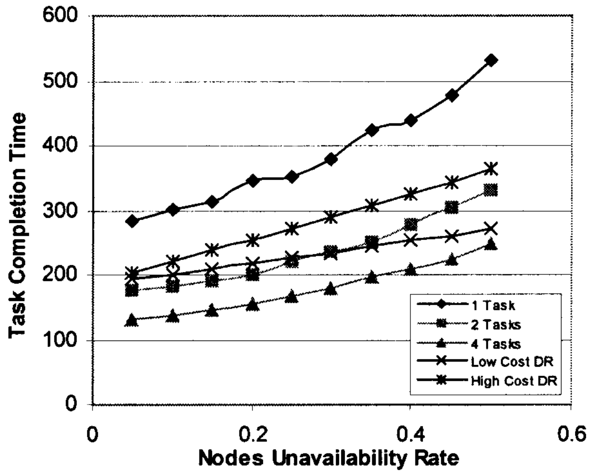

Figure 5: Replicate Scheduling vs. Dynamic Rescheduling on a Computational Grid with Nodes having a Wide Range of Computational Capabilities

5. Computational Replication in Monte Carlo Applications

Among grid applications, those using Monte Carlo methods, which are widely used in scientific computing and simulation, possess good characteristics for the grid-computing environment. Many of these characteristics, such as the statistical nature of Monte Carlo methods and the cryptographic aspects of the underlying random number generators are discussed in [11]. Typically, this paper is interested applying the computational replication technique to grid Monte Carlo applications to improve their performance.

In the typical execution of a Monte Carlo computation on a grid system, we split the entire computational task into N subtasks, with each subtask based on unique independent random number streams. We then schedule each subtask onto the nodes in the grid system. In this case, the assembly of the final result requires all of the N partial results generated from the N subtasks. In this situation, each subtask is a "key" subtask, since the suspension or delay of any one of these subtasks will have a direct effect on the completion time of the whole task. To address this issue, we can use the so-called *N-out-of-M* subtask scheduling strategy specific for grid-based Monte Carlo applications, which is an extension of the computational replication technique.

5.1 The *N-out-of-M* Strategy

To reduce the completion time of the whole Monte Carlo task, we may use the computational replication technique discussed in previous sections by replicating each subtasks. Nevertheless, when we studied the statistical nature of generic Monte Carlo applications, we found that we could take advantage of these characteristics to develop a more efficient way to reduce their task completion time on a computational grid.

When we are running Monte Carlo applications, what we really care about is how many random samples (random trajectories) we must generate to achieve a certain, predetermined, accuracy. We do not much care which random sample set is used, provided that all the random samples are independent in a statistical sense. The statistical nature of Monte Carlo applications allows us to enlarge the actual size of the computation by increasing the number of subtasks from N to M $(M > N)$. Each of these M subtasks uses its unique independent random number set, and we submit M instead of N subtasks to the grid system. Unlike the computational replication technique we discussed in previous sections, where all the replicated tasks are identical, in the *N-out-of-M* strategy, each subtask works with a different random number set. Therefore, M bags of computation will be executed and M partial results may be eventually generated. However, it is not necessary to wait for all M subtasks to finish. When N partial results are ready, we consider the whole Monte Carlo task as completed. The application then collects the N partial results and produces the final result. At this point, the grid-computing system may broadcast abort signals to the nodes that are still computing the remaining subtasks. We call this scheduling strategy *the N-out-of-M strategy*. In

IEEE COMPUTER SOCIETY

the *N-out-of-M strategy* more subtasks than are needed are actually scheduled, therefore, none of these subtasks will become a "key" subtask and we can tolerate at most $M - N$ delayed or halted subtasks.

We model the *N-out-of-M* strategy based on a binomial model in [11]. Assume that the probability of a subtask completing by time t is given by $p(t)$. $p(t)$ describes the aggregate probability over the pool of nodes in the grid. Suppose there are S nodes total in the system, and node i has service rate θ_i. At time t, the probability that a Monte Carlo subtask will be done on node i is $1-e^{\theta_i t}$. Since each node has equal probability to be scheduled a subtask, $p(t)$ can be represented as

$$p(t) = \frac{1}{S}\sum_{i=1}^{S}(1-e^{\theta_i t}) = 1-\frac{1}{S}\sum_{i=1}^{S}e^{\theta_i t}.$$

If $\theta_1, \theta_2, \ldots, \theta_S$ conforms to a distribution with probability density function $\phi(\theta)$, $p(t)$ can thus be written as

$$p(t) = 1-\frac{1}{S}\int_0^L e^{\phi(\theta)t}d\theta.$$

Here L is the maximum value of θ_i in the computation. Typically, if all of the nodes have the same service rate θ, $p(t)$ can be simplified to

$$p(t) = 1-e^{\theta t}.$$

Then, the probability that exactly N out of M subtasks are complete at time t is given by

$$P_{Exactly-N-out-of-M}(t) = \binom{M}{N}p^N(t)\times(1-p(t))^{M-N}.$$

We can approximate $P_{N\text{-}out\text{-}of\text{-}M}(t)$ using a Poisson distribution with $\lambda = N^* p(t)$. Then, $P_{exactly\text{-}N\text{-}out\text{-}of\text{-}M}(t)$ can be approximated as

$$P_{N-out-of-M}(t) \approx \frac{\lambda^M}{M!}e^{-\lambda}.$$

The probability that at least N subtasks are complete is thus given by

$$P_{N-out-of-M}(t) = \sum_{i=N}^{M}\binom{M}{i}p^i(t)\times(1-p(t))^{M-i}.$$

The old strategy can be thought of as "*N-out-of-N*" which has probability given by

$$P_{N-out-of-N}(t) = p^N(t).$$

Now the question is to decide on a reasonable value for M to satisfy a required task completion probability α (when N subtasks are complete on the grid). Unfortunately, it is hard to explicitly represent M in an analytical mode. However, we use a numerical method, which gradually increases M by 1 to evaluate $P_{N\text{-}out\text{-}of\text{-}M}(t)$ until the value of $P_{N\text{-}out\text{-}of\text{-}M}(t)$ is greater than α. This empirically gives us the minimum value of M.

Also notice that the Monte Carlo computation using the *N-out-of-M strategy* is reproducible, because we know exactly which N out of M subtasks are actually involved and which random number streams were used. Thus each of these N subtasks can be reproduced later. However, if we want to reproduce all of these N subtasks at a later time on the computational grid system, the *N-out-of-N* strategy must be used!

5.2 Simulation of the *N-out-of-M* Strategy

Again, we simulate the computational grid's behavior to validate our model of the *N-out-of-M* strategy. In this simulation, we run a Monte Carlo task with 10 subtasks on a 1,000 node computational grid. Figure 6 shows our simulation results and model prediction of the *N-out-of-M* strategy for grid Monte Carlo applications. Again, our analytical model matches the simulation results quite well. Also, we can find that with a proper choice of M (20 in the graph), the Monte Carlo task completion time can be improved significantly over the *N-out-of-N* strategy. However, if we enlarge M too much, the workload of the system increases without significantly reducing the Monte Carlo task completion time.

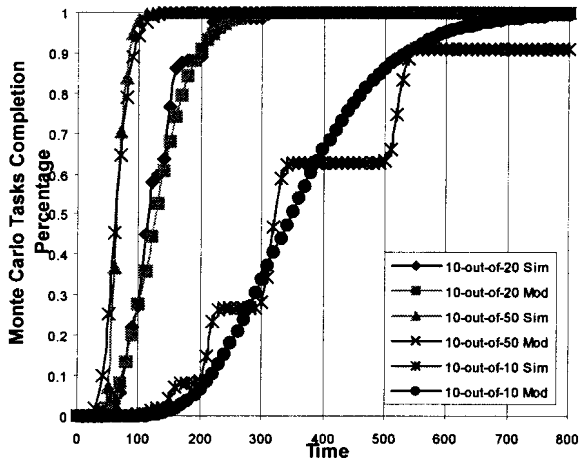

Figure 6: The Comparison of Simulation and Model Prediction of the *N-out-of-M* Scheduling Strategy for Grid Monte Carlo Applications

6. Conclusions

In this paper, we discussed using the computational replication technique to reduce a grid task's completion time and improve a computational grid system's throughput. We established an analytical model of replicate scheduling and simulated the computational grid's behavior to validate it. Our model and simulation results show that using an appropriate number of task replicas can significantly reduce the execution time of a task in the computational grid with high probability. By comparing the performance of dynamic rescheduling with that of computational replication, we conclude that computational replication is effective on a computational grid with nodes having varying performance and slow connections while dynamic rescheduling fits for cluster-like grids with low task rescheduling costs. Also, we extended the computational replication technique to an *N-out-of-M* schedule strategy specifically for grid Monte Carlo applications. Similarly, we found that properly scheduling more subtasks to the grid system will reduce the completion time of the Monte Carlo task effectively.

When we set up the model for replicate scheduling, we did not consider the behavior of the interconnection network. However, in an actual computational grid, the network behavior may seriously affect the task completion time. In our next phase of research, we will improve our model by taking this networking factor into consideration, i.e., considering the message-to-instruction rate in a parallel system with an unbounded number of processors [12]. We also compared the computational replication and the dynamic rescheduling technique in this paper. However, these two techniques can be used together. In the future, we will investigate the performance of such a combined techniques. Also, in the *N-out-of-M* strategy for Monte Carlo applications discussed in this paper, each subtask has the same size in terms of random samples required. However, if information on individual node performance is available, an optimal decomposition into subtasks may require creating subtasks of different sized based on the grid node for execution. While migration under these conditions is questionable, the investigation of the non-uniform partitioning seems warranted in the future.

References

[1] I. Foster, C. Kesselman, and S. Tueske, "The Anatomy of the Grid," International Journal of Supercomputer Applications, **15**(3), 2001.

[2] C. L. Liu, "Deterministic Job Scheduling in Computing System," Modeling and Performance Evaluation of Computer System, North-Holland Publishing Company, 1976.

[3] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling To Minimize Average Completion Time: Off-line and On-line Algorithms," Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithm, pp. 142-151, 1996.

[4] C. Aktouf, O.Benkahla, C.Robach, and A. Guran, "Basic Concepts & Advances in Fault-Tolerant Computing Design," World Scientific Publishing Company, 1998.

[5] L. F. G. Sarmenta, "Sabotage-Tolerance Mechanisms for Volunteer Computing Systems," Proceedings of ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01), Brisbane, Australia, May, 2001.

[6] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, "SETI@home-Massively distributed computing for SETI," Computing in Science and Engineering, v3n1, 81, 2001.

[7] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities," Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2002, 2002.

[8] A. Baratloo, M. Karaul, Z. Kedem, P. Wyckoff, "Charlotte: Metacomputing on the Web," Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems, 1996.

[9] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," Proceedings of the 8th International Conference of Distributed Computing Systems, pp. 104-111, 1988.

[10] Entropia website, http://www.entropia.com.

[11] Y. Li and M. Mascagni, "Grid-based Monte Carlo Applications," Lecture Notes in Computer Science, **2536**: 13-24, Proceedings of the International Workshop/Conference on Grid Computing, GRID2002, Baltimore, 2002.

[12] C. H. Papadimitriou and M. Yannakakis, "Towards an Architecture-Independent Analysis of Parallel Algorithms," SIAM Journal on Computing, 19(2):322-328, 1990.

[13] Condor website, http://www.cs.wisc.edu/condor.

IEEE COMPUTER SOCIETY