Computational Infrastructure for Parallel, Distributed, and Grid-based Monte Carlo Computations

Michael Mascagni and Yaohang Li

Department of Computer Science *and* School of Computational Science and Information Technology, Florida State University, Tallahassee, FL 32306-4530 **USA**
E-mail: Michael.Mascagni@fsu.edu, yaohang@csit.fsu.edu,
Home page: http://www.cs.fsu.edu/~mascagni

Abstract. Monte Carlo applications are widely perceived as computationally intensive but naturally parallel. Therefore, they can be effectively executed using the dynamic bag-of-work model which is well suited to parallel, distributed, and grid-based architectures. This paper concentrates on providing computational infrastructure for Monte Carlo applications on such architectures. This is accomplished by analyzing the characteristics of large-scale Monte Carlo computations, and leveraging the existing Scalable Parallel Random Number Generators (SPRNG) library. Based on these analyses, we improve the efficiency of subtask-scheduling by implementing and analyzing the "N-out-of-M" strategy, and develop a Monte Carlo-specific lightweight checkpointing technique, which leads to a performance improvement. Also, we enhance the trustworthiness of Monte Carlo applications on these architectures by utilizing the statistical nature of Monte Carlo and by cryptographically validating intermediate results utilizing the random number generator already in use in the Monte Carlo application. All these techniques lead to a high-performance grid-computing infrastructure that is capable of providing trustworthy Monte Carlo computation services.

1 Introduction

Readers are most likely already familiar with parallel and distributed computing architectures, so we choose not to elaborate on them further. Thus, grid computing is characterized by the large-scale sharing and cooperation of dynamically distributed resources, such as CPU cycles, communication bandwidth, and data, to constitute a computational environment [1]. In the grid's dynamic environment, from the application point-of-view, two issues are of prime importance: performance – how quickly the grid-computing system can complete the submitted tasks, and trustworthiness – that the results obtained are, in fact, due to the computation requested. To meet these two requirements, many grid-computing or distributed-computing systems, such as Condor [2], HARNESS [3], Javelin [4], Globus [5], and Entropia [7], concentrate on developing high-performance and trust-computing facilities through system-level approaches. In this paper, we are

going to analyze the characteristics of Monte Carlo applications, which are a potentially large computational category of parallel, distributed, and grid applications, to develop approaches to address performance and trustworthiness issues at the application level.

The remainder of this paper is organized as follows. In §2, we analyze the characteristics of Monte Carlo applications. We also quickly describe the random number generation requirements for these applications on parallel, distributed, and grid-based architectures. We then develop a generic grid-computing paradigm for Monte Carlo computations. We discuss how to take advantage of the characteristics of Monte Carlo applications to improve the performance and trustworthiness of Monte Carlo grid computing in §3 and §4, respectively. Finally, §5 summarizes our conclusions and future research directions.

2 Grid-based Monte Carlo Applications

Among parallel, distributed, and grid applications, those using Monte Carlo methods, which are widely used in scientific computing and simulation, have been considered too simplistic for consideration due to their natural parallelism. However, below we will show that many aspects of Monte Carlo applications can be exploited to provide much higher levels of performance and trustworthiness for computations on these architectures. According to word of mouth, about 50% of the CPU time used on supercomputers at the U. S. Department of Energy National Labs is spent on Monte Carlo computations. Unlike data-intensive applications, Monte Carlo applications are usually computation intensive [6], and they tend to work on relatively small data sets. Parallelism is a way to accelerate the convergence of a Monte Carlo computation. If N processors execute N independent copies of a Monte Carlo computation, the accumulated result will have a variance N time smaller than that of a single copy. In a distributed Monte Carlo application, once a distributed task starts, it can usually be executed independently with almost no interprocess communication. Therefore, Monte Carlo applications are perceived as naturally parallel, and they can usually be programmed via the so-called dynamic *bag-of-work* model. Here a large task is split into smaller independent subtasks and each are then executed separately. Effectively using the dynamic *bag-of-work* model for Monte Carlo requires that the underlying random number streams in each subtask be independent in a statistical sense. The `SPRNG` (Scalable Parallel Random Number Generators) library [11] was designed to use parameterized pseudorandom number generators to provide independent random number streams to parallel processes. Some generators in `SPRNG` can generate up to 2^{78000} - 1 independent random number streams with sufficiently long period and good quality [13]. These generators meet the random number requirements of most Monte Carlo applications for these types of architectures.

`SPRNG` was originally designed to provide independent and dynamic streams of random numbers for massively parallel supercomputers. The design of `SPRNG` was based on the needs of Monte Carlo applications, like neutronics. In neutronics, it

is commonplace to associate a single random number stream with each neutron. When a neutron splits into more neutrons, each child neutron is then associated with a new (and hopefully independent) random number stream. Given such a computation, `SPRNG` ensures that, when desired, such a computation performed on a parallel or distributed machine can be entirely reproducible. The details of how this is achieved is beyond the current scope [11]. However, these capabilities of `SPRNG` are extensible to distributed and grid-based architectures, and form the underpinnings for the grid services we will discuss. In the remainder of this paper we thus focus exclusively on grid computing issues for Monte Carlo applications.

The intrinsically parallel aspect of Monte Carlo applications makes them an ideal fit for the grid-computing paradigm. In general, grid-based Monte Carlo applications can divide the Monte Carlo task into a number of subtasks by the *task-split service* and utilize the grid's *schedule service* to dispatch these independent subtasks to different nodes [15]. The *connectivity services* provide communication facilities among nodes providing *computational services*. The execution of a subtask takes advantage of the *storage service* of the grid to store intermediate results and to store each subtask's final (partial) result. When the subtasks are done, the *collection service* can be used to gather the results and generate the final result of the entire computation.

The inherent characteristics of Monte Carlo applications motivate the use of grid computing to effectively perform large-scale Monte Carlo computations. Furthermore, within this Monte Carlo grid-computing paradigm, we can use the statistical nature of Monte Carlo computations and the cryptographic aspects of random numbers to reduce the wallclock time and to enforce the trustworthiness of the computation.

3 Improving the Performance of Grid-based Monte Carlo Computing

3.1 The N-out-of-M Strategy

Subtask-Scheduling using the N-out-of-M Strategy The nodes that provide CPU cycles in a grid system will most likely have computational capabilities that vary greatly. A node might be a high-end supercomputer, or a low-end personal computer, even just an intelligent widget. In addition, these nodes are geographically widely distributed and not centrally manageable. A node may go down or become inaccessible without notice while it is working on its task. Therefore, a slow node might become the bottleneck of the whole computation if the assembly of the final result must wait for the partial result generated on this slow node. A delayed subtask might delay the accomplishment of the whole task while a halted subtask might prevent the whole task from ever finishing. To address this problem, system-level methods are used in many grid systems. For example, `Entropia` [7] tracks the execution of each subtask to make sure none of the subtasks are halted or delayed. However, the statistical nature of Monte Carlo applications provides a shortcut to solve this problem at the application level.

Suppose we are going to execute a Monte Carlo computation on a grid system. We split it into N subtasks, with each subtask based on its unique independent random number stream. We then schedule each subtask onto the nodes in the grid system. In this case, the assembly of the final result requires all the N partial results generated from the N subtasks. 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.

When we are running Monte Carlo applications, what we really care about is how many random samples (random trajectories) we must obtain to achieve a certain, predetermined, accuracy. We do not much care which random sample set is estimated, 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, where $M > N$. Each of these M subtasks uses its unique independent random number set, and we submit M instead of N subtasks to the grid system. Therefore, M bags of computation will be carried out 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 task for the grid system to be 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 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.

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 numbers 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 grid system, the *N-out-of-N* strategy must be used.

One drawback of the *N-out-of-M* strategy is we must execute more subtasks than actually needed and will therefore increase the computational workload on the grid system. However, our experience with distributed computing systems such as `Condor` and `Javelin` shows that most of the time there are more nodes providing computing services available in the grid system than subtasks. Therefore, properly increasing the computational workload to achieve a shorter completion time for a computational task should be an acceptable tradeoff in a grid system.

Analysis of the N-out-of-M Strategy In Monte Carlo applications, N is determined by the application and depends on the number of random samples or random trajectories needed to obtain a predetermined accuracy. The problem is thus how to choose the value M properly. A good choice of M can prevent a few subtasks from delaying or even halting the whole computation. However, if M is chosen too large, there may be little benefit to the computation at

the cost of significantly increasing the workload of the grid system. In order to determine a proper value of M to achieve a specific performance requirement, we study the grid behavior and consider some system parameters. In the *N-out-of-M* strategy, the completion time of a Monte Carlo computational task depends on the performance of each individual node that is assigned a subtask, the node failure rate, and also the interconnection 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, the tasks' scheduling time and result collection time 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.

To analyze this we establish a Petri Net (PN) model of the *N-out-of-M* strategy. This PN model has M nodes in total. 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 subtask completion condition (token in $p_{subtask}$) is reached when W is hit. When $p_{subtask}$ gathers N tokens, transition $t_{N-out-of-M}$ enables firing and a token in $p_{complete}$ indicates the completion of the Monte Carlo task.

We also establish a simpler binomial model for the subtask-scheduling scheme using the *N-out-of-M* strategy based on the above PN model. Assume that the probability of a subtask completing by time t is given by $p(t)$. The function $p(t)$ describes the aggregate probability over the pool of nodes in the grid. In a real-life grid system, $p(t)$ could be measured by computing the empirical frequencies of completion times over the pool. In this paper, we model $p(t)$ based on an analytic probability distribution function.

Let S be the total number of nodes available in the grid system,

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

$\theta_i\prime$ 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 node availability, the actual service rate, θ_i, in node i is $\theta_i = \theta'_i * p^i_{sys}$.

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

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

If the service rates, θ_1, θ_2, ..., θ_S, conform to a distribution with probability density function $\phi(\theta)$, $p(t)$ can thus be written as

$$p(t) = 1 - \frac{1}{L}\int_0^L e^{\phi(\theta)t} d\theta. \tag{2}$$

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

$$p(t) = 1 - e^{\theta\, t}. \tag{3}$$

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

$$P_{Exactly-N-out-of-M}(t) = \binom{M}{N} p^N(t) \times (1 - p(t))^{M-N}. \tag{4}$$

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

$$P_{Exactly-N-out-of-M}(t) \approx \frac{\lambda^M}{M!} e^{-\lambda}. \tag{5}$$

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

$$P_{N-out-of-M}(t) = \sum_{i=N}^{M} \binom{M}{i} p^i(t) \times (1 - p(t))^{M-i}. \tag{6}$$

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

$$P_{N-out-of-N}(t) = p^N(t). \tag{7}$$

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 analytic form. However, we use a numerical method, which gradually increases M by 1 to evaluate $P_{N-out-of-M}(t)$ until the value of $P_{N-out-of-M}(t)$ is greater than α. This empirically gives us the minimum value of M. An alternative approach to estimate M/N is to use a normal distribution to approximate the underlying binomial. When $M * (1 - p(t)) \geq 5$ and $M * p(t) \geq 5$, the binomial distribution can be approximated by a normal curve with mean $m = M * p(t)$ and standard deviation $\sigma = \sqrt{Mp(t)(1 - p(t))}$. Then, we can find the minimum value M that satisfies

$$\Phi(\frac{M - m}{\sigma}) - \Phi(\frac{N - m}{\sigma}) \geq \alpha, \tag{8}$$

where Φ is the normal cumulative distribution function.

In a grid system, nodes providing computational services join and leave dynamically. Some nodes are considered to be "transient" nodes, which provide

computational services temporarily and may depart from the system permanently. A subtask submitted to a "transient" node may have no chance of being finished. Suppose the fraction of "transient" nodes in a grid is β, then, we need to enlarge M to $\lceil M/(1-\beta) \rceil$ to tolerate these never-finished subtasks.

Simulation of the N-out-of-M Strategy In our simulation program of the *N-out-of-M* strategy, 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 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 performance of the *N-out-of-M* strategy in grid systems with different configurations.

Figure 1 shows our simulation results and model prediction of the *N-out-of-M* strategy for grid Monte Carlo applications. Our analytical model matches the simulation results quite well. Also, we can find that with the 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. Also, we notice that as time goes on, the *N-out-of-M* strategy always has a higher probability of completion than the *N-out-of-N* strategy, although they all converge to probability one at large times.

Figures 2 and 3 show the simulation results of the *N-out-of-M* strategy in different grid systems. Both simulated grid systems assume that the service rates θ of nodes are normally distributed with the same means (0.005) but different variances (0.001 in Figure 2 and 0.003 in Figure 3). Figure 2 simulates a 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. On the other hand, Figure 3 is the simulation of a 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` [9], where a node might be a high-end supercomputer, or a low-end personal computer. From the graphs, we see that the *N-out-of-M* scheduling strategy improves the Monte Carlo task completion time in both grid systems; however, we gain more significant improvement in the system comprised of nodes with service rates having a large variance. This experimental result indicates that the *N-out-of-M* strategy is more effective in a grid system where an individual node's performance varies greatly. More interestingly, the simulation results also show that, in both grid systems, with a sufficiently large value of M, the time values after which the Monte Carlo task is complete with a high probability is close to 200 time steps,

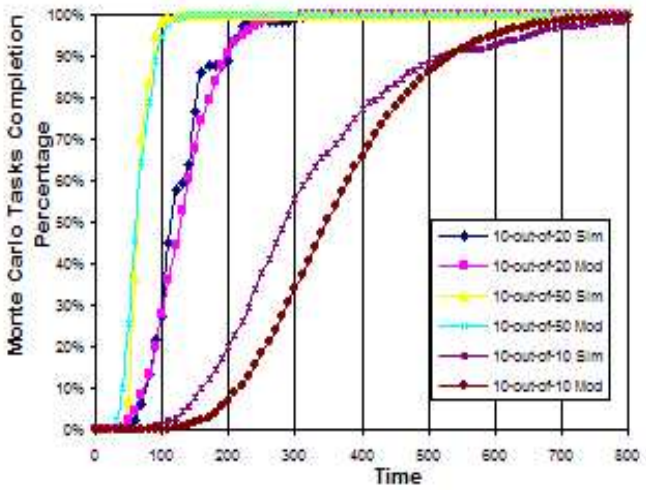

Fig. 1. Simulations and Model Prediction of the *N-out-of-M* Scheduling Strategy for Grid Monte Carlo Applications.

which is exactly the subtask completion time for a single node with the mean (0.005) service rate. Therefore, we can expect that, with a proper number of subtasks scheduled using the *N-out-of-M* strategy, the Monte Carlo task completion time on a grid can be made to be almost the same as the subtask completion time in a node with average computational capability.

Lightweight Checkpointing A subtask running on a node of a grid system may take a very long time to finish. The *N-out-of-M* strategy is an attempt to mitigate the effect of this on the overall running time. However, if checkpointing is incorporated, one can directly attack reducing the completion time of the subtasks. Some grid computing systems implement a process-level checkpoint. `Condor`, for example, takes a snapshot of the process's current state, including stack and data segments, shared library code, process address space, all CPU states, states of all open files, all signal handlers, and pending signals [12]. On recovery, the process reads the checkpoint file and then restores its state. Since the process state contains a large amount of data, processing such a checkpoint is quite costly. Also, process-level checkpointing is very platform-dependent, which limits the possibility of migrating the process-level checkpoint to another node in a heterogeneous grid-computing environment.

Fortunately, Monte Carlo applications have a structure highly amenable to application-based checkpointing. Typically, a Monte Carlo application starts in an initial configuration, evaluates a random sample or a random trajectory, estimates a result, accumulates means and variances with previous results, and repeats this process until some termination condition is met. Although differ-

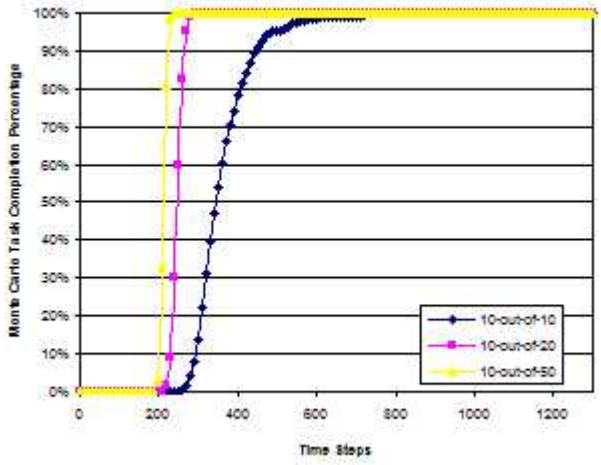

Fig. 2. Simulations of the *N-out-of-M* Strategy on a Grid System with Nodes Service Rates Normally Distributed (Mean=0.005, Variance=0.001).

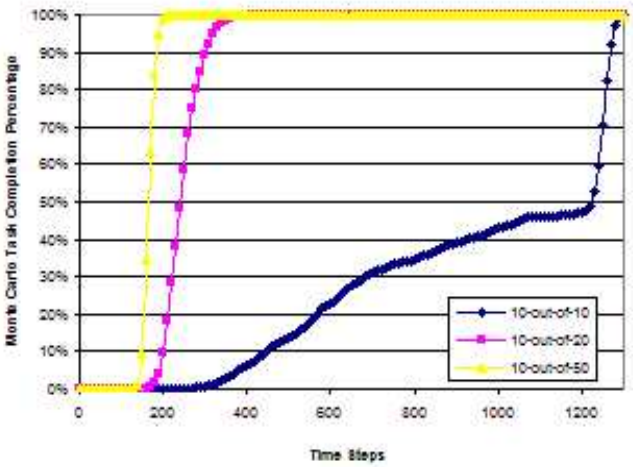

Fig. 3. Simulations of the *N-out-of-M* Strategy on a Grid System with Nodes Service Rates Normally Distributed (Mean=0.005, Variance=0.003).

ent Monte Carlo applications may have very different implementations, many of them can be developed or adjusted in a typical programming structure that consists of initialization, followed by a loop for generating statistical samples, and ending with the summation of the overall statistics.

Thus, to recover an interrupted computation, a Monte Carlo application needs to save only a relatively small amount of information. The necessary information to reconstruct a Monte Carlo computation image at checkpoint time will be the current results based on the estimates obtained so far, the current status and parameters of the random number generators, and other relevant program information like the current iteration number. This allows one to make a smart and quick application checkpoint in most Monte Carlo applications. Using `XML` [8] to record the checkpointing information, we can make this checkpoint platform-independent. More importantly, compared to a process checkpoint, the application-level checkpoint is much smaller in size and much quicker to generate. Therefore, it should be relatively easy to migrate a Monte Carlo computation from one node to another in a grid system. With the application-level checkpointing and recovery facilities, the typical Monte Carlo application's programming structure can be amended as described above. However, the implementation of application level checkpointing will somewhat increase the complexity of developing new Monte Carlo grid applications.

4 Enhancing the Trustworthiness of Grid-based Monte Carlo Computing

4.1 Distributed Monte Carlo Partial Result Validation

The correctness and accuracy of grid-based computations are vitally important to an application. In a grid-computing environment, the service providers of the grid are often geographically separated with no central management. Faults may hurt the integrity of a computation. These might include faults arising from the network, system software or node hardware. A node providing CPU cycles might not be trustworthy. A user might provide a system to the grid without the intent of faithfully executing the applications obtained. Experience with `SETIhome` has shown that users often fake computations and return wrong or inaccurate results. The resources in a grid system are so widely distributed that it appears difficult for a grid-computing system to completely prevent all "bad" nodes from participating in a grid computation. Unfortunately, Monte Carlo applications are very sensitive to each partial result generated from each subtask. An erroneous partial result will most likely lead to the corruption of the whole grid computation and thus render it useless.

To enforce the correctness of the computation, many distributed computing or grid systems adapt fault-tolerant methods, like duplicate checking [10] and majority vote [16]. In these approaches, subtasks are duplicated and carried out on different nodes. Erroneous partial results can be found by comparing the partial results of the same subtask executed on different nodes. Duplicated checking

requires doubling computations to discover an erroneous partial result. Majority vote requires at least three times more computation to identify an erroneous partial result. Using duplicate checking or majority vote will significantly increase the workload of a grid system.

In the dynamic *bag-of-work* model as applied to Monte Carlo applications, each subtask works on the same description of the problem but estimates based on different random samples. Since the mean in a Monte Carlo computation is accumulated from many samples, its distribution will be approximately normal based in the Central Limit Theorem. Suppose $f_1, \ldots, f_i, \ldots, f_n$ are the n partial results generated from individual nodes on a grid system. The mean of these partial results is

$$\hat{f} = \frac{1}{n} \sum_{i=1}^{n} f_i, \tag{9}$$

and we can estimate its standard error, s, via the following formula

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (f_i - \hat{f})^2}. \tag{10}$$

Specifically, the Central Limit Theorem states that $\hat{f}$ should be distributed approximately as a Student-t random variable with mean $\hat{f}$, standard deviation $s/\sqrt{n}$, and n degrees-of-freedom. However, since n, the number of subtasks, is often large, we may instead approximate the Student-t distribution with the normal. Standard normal confidence interval theory states that with 68% confidence that the exact mean is within 1 standard deviation of $\hat{f}$, with 95% confidence within 2 standard deviations, and 99% confidence within 3 standard deviations. This statistical property of Monte Carlo computation can be used to develop an approach for validating the partial results of a large grid-based Monte Carlo computation.

Here is the proposed method for distributed Monte Carlo partial result validation. Suppose we are running n Monte Carlo subtasks on the grid, and the ith subtask returns partial result, f_i. We anticipate that the f_i are approximately normally distributed with mean, $\hat{f}$, and standard deviation, $\sigma = s/\sqrt{n}$. We expect that about one of the f_i in this group of n to lie outside a normal confidence interval with confidence $1 - 1/n$. In order to choose a confidence level that permits events we expect to see, statistically, yet flags events as outliers requires us to choose a multiplier, c, so that we flag events that should only occur once in a group of size cn. The choice of c is rather subjective, but $c = 10$ implies that in only 1 in 10 runs of size n we should expect to find an outlier with confidence $1 - 1/10n$. With a given choice of c, one computes the symmetric normal confidence interval based on a confidence of $\alpha\% = 1 - 1/cn$. Thus the confidence interval is $[\hat{f} - Z_{\alpha/2}\sigma,\ \hat{f} + Z_{\alpha/2}\sigma]$, where $Z_{\alpha/2}$ is unit normal value such that $\int_0^{Z_{\alpha/2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\, dx = \frac{\alpha}{2}$. If f_i is in this confidence interval, we can consider this

partial result as trustworthy. However, if f_i falls out of the interval, which may happen merely by chance with a very small probability, this particular partial result is suspected.

There are two possibilities for a partial result, f_i, to fall out of the confidence interval. These are

1. errors occur during the computation of this subtask, or
2. a rare event with very low probability is captured.

In former case, this partial result is erroneous and should be discarded, whereas in the latter case, we need to take it into consideration. To identify these two cases, we can rerun the particular subtask that generated the suspicious partial result on a trusted node for further validation.

This Monte Carlo partial result validation method supplies us with a way to identify suspicious results without running more subtasks. This method assumes that the majority of the nodes in grid system are "good" service providers, which can correctly and faithfully execute their assigned task and transfer the result. If most of the nodes are malicious, this validation method may not be effective. However, experience has shown that the fraction of "bad" nodes in volunteered computation is very small.

4.2 Intermediate Value Checking

Usually, a grid-computing system compensates the service providers to encourage computer owners to supply resources. Many Internet-wide grid-computing projects, such as `SETI@home` [9], have the experience that some service providers don't faithfully execute their assigned subtasks. Instead they attempt to provide bogus partial results at a much lower personal computational cost in order to obtain more benefits. Checking whether the assigned subtask from a service provider is faithfully carried out and accurately executed is a critical issue that must be addressed by a grid-computing system.

One approach to check the validity of a subtask computation is to validate intermediate values within the computation. Intermediate values are quantities generated within the execution of the subtask. To the node that runs the subtask, these values will be unknown until the subtask is actually executed and reaches a specific point within the program. On the other hand, to the clever application owner, certain intermediate values are either pre-known and secret or are very easy to generate. Therefore, by comparing the intermediate values and the pre-known values, we can control whether the subtask is actually faithfully carried out or not. Monte Carlo applications consume pseudorandom numbers, which are generated deterministically from a pseudorandom number generator. If this pseudorandom number generator has a cheap algorithm for computing arbitrarily within the period, the random numbers are perfect candidates to be these cleverly chosen intermediate values. Thus, we have a very simple strategy to validate a result from subtasks by tracing certain predetermined random numbers in Monte Carlo applications.

For example, in a grid Monte Carlo application, we might force each subtask to save the value of the current pseudorandom number after every N (e.g., $N =$ 100,000) pseudorandom numbers are generated. Therefore, we can keep a record of the Nth, $2N$th, ..., kNth random numbers used in the subtask. To validate the actual execution of a subtask on the server side, we can just recompute the Nth, $2N$th, ..., kNth random numbers applying the specific generator with the same seed and parameters as used in this subtask. We then simply match them. A mismatch indicates problems during the execution of the task. Also, we can use intermediate values of the computation along with random numbers to create a cryptographic digest of the computation in order to make it even harder to fake a computational result. Given our list of random numbers, or a deterministic way to produce such a list, when those random numbers are computed, we can save some piece of program data current at that time into an array. At the same time we can use that random number to encrypt the saved data and incorporate these encrypted values in a cryptographic digest of the entire computation. At the end of the computation the digest and the saved values are then both returned to the server. The server, through cryptographic exchange, can recover the list of encrypted program data and quickly compute the random numbers used to encrypt them. Thus, the server can decrypt the list and compare it to the "plaintext" versions of the same transmitted from the application. Any discrepancies would flag either an erroneous or faked result. While this technique is certainly not a perfect way to ensure correctness and trustworthiness, a user determined on faking results would have to scrupulously analyze the code to determine the technique being used, and would have to know enough about the mathematics of the random number generator to leap ahead as required. In our estimation, surmounting these difficulties would far surpass the amount of work saved by gaining the ability to pass off faked results as genuine.

5 Conclusions

Monte Carlo applications generically exhibit naturally parallel and computationally intensive characteristics. Moreover, we can easily fit the dynamic *bag-of-work* model, which works so well for Monte Carlo applications, onto a grid system to implement large-scale grid-based Monte Carlo computing. Furthermore, based on the analysis of grid-based Monte Carlo applications, we may take advantage of the statistical nature of Monte Carlo calculations and the cryptographic nature of random numbers to enhance the performance and trustworthiness of this Monte Carlo grid-computing infrastructure at the application level.

We are developing a Grid-Computing Infrastructure for Monte Carlo Applications (GCIMCA) based on the Globus toolkit [5] and the `SPRNG` library [11], using the techniques described in this paper. The infrastructure software aims to provide grid services to facilitate the development of grid-based Monte Carlo applications and the execution of large-scale Monte Carlo computations in a grid-computing environment. At the same time, we are also trying to execute some real-life large-scale Monte Carlo applications, such as the Monte Carlo sim-

ulation of Ligand-Receptor interaction in structured protein systems [17] and the Monte Carlo molecular modeling applications, on our developing grid-computing infrastructure.

References

1. I. Foster, C. Kesselman, and S. Tueske, "The Anatomy of the Grid," *International Journal of Supercomputer Applications*, **15**(3): 1–25, 2001.
2. M. Litzkow, M. Livny, and M. Mutka, "Condor–A Hunter of Idle Workstations," *Proceedings of the 8th International Conference of Distributed Computing Systems*: 104–111, 1998.
3. M. Beck, J. Dongarra, G. Fagg, A. Geist, P. Gray, J. Kohl, M. Migliardi, K. Moore, T. Moore, P. Papadopoulous, S. Scott, and V. Sunderam, "HARNESS: A Next Generation Distributed Virtual Machine," *Journal of Future Generation Computer Systems*, **15**(5/6): 571–582, 1999.
4. B. O.Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu, "Javelin: Internet-Based Parallel Computing Using Java," *Concurrency: Practice and Experience*, **9**(11): 1139–1160, 1997.
5. I. Foster and C. Kesselman, "Globus: A Mmetacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, **11**(2): 115–128, 1997.
6. A. Srinivasan, D. M. Ceperley, and M. Mascagni, "Random Number Generators for Parallel Applications," *Monte Carlo Methods in Chemical Physics*, **105**: 13-36, 1997.
7. Entropia website: `http://www.entropia.com`.
8. XML website: `http://www.xml.org`.
9. E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, "SETI@home-Massively Distributed Computing for SETI," *Computing in Science and Engineering*, **3**(1): 78–83, 2001.
10. C. Aktouf, O. Benkahla, C. Robach, and A. Guran, *Basic Concepts & Advances in Fault-Tolerant Computing Design*, World Scientific Publishing Company, 1998.
11. M. Mascagni and A. Srinivasan, "Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation," *ACM Transactions on Mathematical Software*, **26**: 436–461, 2000.
12. M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for High Throughput Computing," *SPEEDUP Journal*, **11**(1), 1997.
13. `SPRNG` website: `http://sprng.cs.fsu.edu`.
14. `Condor` website: `http://www.cs.wisc.edu/condor`.
15. R. Buyya, S. Chapin, and D. DiNucci, "Architectural Models for Resource Management in the Grid," *Proceedings of The First IEEE/ACM International Workshop on Grid Computing (GRID2000), Springer Verlag LNCS Series*, 2000.
16. 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)*, 2001.
17. Y. Li, M. Mascagni, and M. H. Peters, "Grid-based Nonequilibrium Multiple-Time Scale Molecular Dynamics/Brownian Dynamics Simulations of Ligand-Receptor Interactions in Structured Protein Systems," *Proceedings of the First BioGrid Workshop at the 3rd IEEE/ACM Symposium Cluster Computing and the Grid*, 2003.