Monte Carlo Methods and Appl., Vol. 11, No. 1, pp. 39 – 55 (2005)
© *VSP* 2005

Grid-based Quasi-Monte Carlo Applications

Yaohang Li[1] **and Michael Mascagni**[2]

[1]Department of Computer Science, North Carolina A&T State University, Greensboro, NC 27411 **USA**, yaohang@ncat.edu

[2]Department of Computer Science and School of Computational Science, Florida State University, Tallahassee, FL 32306-4560 **USA**, mascagni@fsu.edu

Abstracts — In this paper, we extend the techniques used in Grid-based Monte Carlo applications to Grid-based quasi-Monte Carlo applications. These techniques include an *N-out-of-M* strategy for efficiently scheduling subtasks on the Grid, lightweight checkpointing for Grid subtask status recovery, a partial result validation scheme to verify the correctness of each individual partial result, and an intermediate result checking scheme to enforce the faithful execution of each subtask. Our analysis shows that the extremely high uniformity seen in quasirandom sequences prevents us from applying many of our Grid-based Monte Carlo techniques to Grid-based quasi-Monte Carlo applications. However, the use of scrambled quasirandom sequence becomes a key to tackling this problem, and makes many of the techniques we used in Grid-based Monte Carlo applications effective in Grid-based quasi-Monte Carlo applications. All the techniques we will describe here contribute to performance improvement and trustworthiness enhancement for large-scale quasi-Monte Carlo applications on the Grid, which eventually lead to a high-performance Grid-computing infrastructure that is capable of providing trustworthy quasi-Monte Carlo computation services.

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 [12]. In the Grid's dynamic environment, from the application point-of-view, two issues are of prime import: 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 [17], HARNESS [6], Javelin [20], Globus [11], and Entropia [2], concentrate on developing high-performance and trust-computing facilities through system-level approaches. In [14], we analyzed the characteristics of Monte Carlo applications to develop approaches to address the performance and trustworthiness issues from the application level. The approaches including the *N-out-of-M* subtask scheduling strategy, lightweight checkpointing, partial result validation, and intermediate value checking, are effective in Grid-based Monte Carlo applications. In this article, we extend these approaches to Grid-based quasi-Monte Carlo applications, which are closely related to Monte Carlo applications, but use quasirandom instead of pseudorandom number. These approaches lead us to develop a high-performance Grid-computing infrastructure that is capable of providing trustworthy quasi-Monte Carlo computational services.

The remainder of this paper is organized as follows. In §2, we compare the characteristics of Monte Carlo applications with that of quasi-Monte Carlo applications for the purpose of Grid computing. We discuss how to extend the techniques used in Grid-based Monte Carlo applications to improve the performance and trustworthiness of Grid-based quasi-Monte Carlo applications in §3 and §4, respectively. Finally, §5 summarizes our results, provides conclusions, and provides discussion of future research directions.

2. Grid-based Monte Carlo Applications and Grid-based Quasi-Monte Carlo Applications

Monte Carlo applications are perceived as computationally intensive and naturally parallel, and they can usually be implemented via the so-called dynamic*bag-of-work* model. In a dynamic *bag-of-work* model using in a Monte Carlo application, a large task is split into smaller independent subtasks, and each are then executed separately. Then, the partial results are collected and used to assemble an accumulated result with smaller variance than that of a single copy. The inherent characteristics of Monte Carlo applications and the dynamic *bag-of-work* model make them a natural fit for the Grid-computing environment.

Quasi-Monte Carlo applications are very similar to Monte Carlo applications. They use deterministic low-discrepancy quasirandom sequences to obtain typically higher convergence rates, $O(N^{-1}(\log N)^k)$, than that of Monte Carlo methods, which is approximately $O(N^{-1/2})$ based on the use of pseudorandom numbers. Discrepancy, which will be described in greater detail below, is a measure the lack of uniformity of a sequence. Thus a low-discrepancy sequence is more uniformly distributed than a pseudorandom number sequence. Similar to Monte Carlo applications, quasi-Monte Carlo applications can also be implemented with the dynamic *bag-of-work* model, which motivates us to adapt quasi-Monte Carlo applications to the Grid to take advantage of the computational power of Grid computing. Figure 1 shows a generic diagram of Grid-based quasi-Monte Carlo application on the Grid using the dynamic *bag-of-work* model.

Effectively using the dynamic *bag-of-work* model for Monte Carlo applications on the

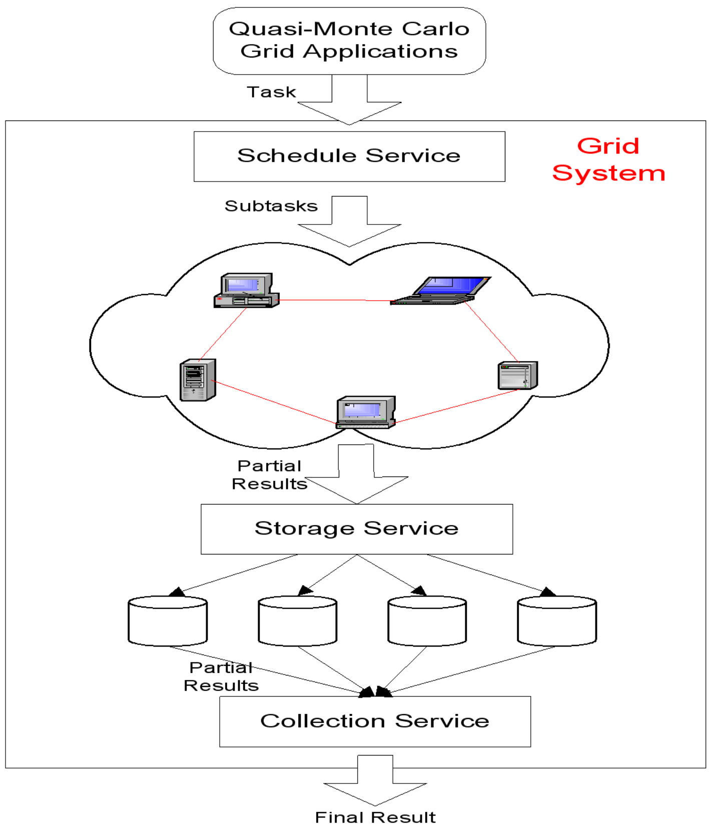

Figure 1: A Generic Diagram of Grid-Based Quasi-Monte Carlo Application on the Grid Using the Dynamic *bag-of-work* model.

Grid requires that the underlying random number streams in each subtask be independent in a statistical sense. Different from Monte Carlo, to take advantage of parallelism in quasi-Monte Carlo, we expect the underlying quasirandom number streams used on each Grid node to have a small discrepancy. Leapfrogging and blocking are two common schemes for parallel quasirandom number generation [7]. However, recent research into scrambled digital quasirandom sequences can be applied to generate large-scale parallel quasirandom sequences with each sequence retaining low-discrepancy properties with some randomness, which enables one to use statistical methods for practical error estimation [10]. Furthermore, the characteristics of scrambled quasirandom sequences provide us a way to extend the techniques used in Grid-based Monte Carlo applications, such as the *N-out-of-M* strategy, partial result validation, and intermediate value checking, to Grid-based quasi-Monte Carlo applications. These techniques can be applied to reduce the wallclock time of a computation and to enforce the trustworthiness of the Grid-based quasi-Monte Carlo computation.

3. Improving the Performance of Grid-based quasi-Monte Carlo Computing

3.1 N-out-of-M Scheduling Strategy

In a Grid-computing environment, the participating computational nodes are usually geographically widely distributed, with various computational capabilities, and not centrally manageable. 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 of tracking each subtask can be used. However, the statistical nature of Monte Carlo applications provides a shortcut to use an *N-out-of-M* subtask scheduling strategy to solve this problem at the application level.

In Grid-based Monte Carlo applications, by using the *N-out-of-M* scheduling strategy, we enlarge the actual size of the computation by increasing the number of subtasks from original N subtasks to Msubtasks, where $M > N$. Each of these M new 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, 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 accumulated result. More analysis of the *N-out-of-M* strategy can be found in [13].

It should be noted that there are many stochastic computations where the *N-out-of-M* strategy could be seriously flawed and highly biased. For example, suppose we were simulating the distribution of the occurrence times of an event in a stochastic process where the calculation time is proportional to the simulated time. The *N-out-of-M* strategy would only simulate the shortest occurrence times, and would give a wildly biased estimate of the distribution. Clearly, this sort of computation is ill-suited to the *N-out-of-M* strategy. In this paper we exclude such computations from consideration, and the examples that we use are expected to suffer no introduction of short computational bias for the use of the *N-out-of-M* strategy, like high-dimensional numerical integration.

The *N-out-of-M* subtask schedule strategy in Grid-based Monte Carlo applications requires the statistical independence of parallel random sequences used in all M subtasks. In contrast, quasi-Monte Carlo applications use highly correlated and uniform quasirandom numbers. Therefore, to apply the *N-out-of-M* subtask schedule strategy to quasi-Monte Carlo applications, the combination of any N out of M parallel quasirandom sequences must remain highly uniform, i. e., the union of the N out of M quasirandom sequences must still have low discrepancy. Intuitively, we may hope that the combination of two low-discrepancy sequences will always lead to a lower discrepancy. The ideal situation is, if two sequences, S_1 and S_2, both contain N quasirandom numbers with discrepancies of $O(N^{-1})$, then the sequence S that is combined from S_1 and S_2 will have discrepancy of $O((2N)^{-1})$. Unfortunately, this is not always true. The simplest counterexample is the combination of two identical low-discrepancy sequences, which will have the same discrepancy as a single copy of the sequence.

The above discussion leads to an interesting question – what is the error bound of a quasi-Monte Carlo computation using parallel quasirandom number sequences? The Koksma-Hlawka inequality [9] is the foundation for analyzing quasi-Monte Carlo integration error. Based on the Koksma-Hlawka inequality, we deduce Lemma 1 that provides an upper bound on the error for a parallel quasi-Monte Carlo integration using multiple low-discrepancy sequences.

Theorem 1 (Koksma-Hlawka Theorem). For any sequence $X = \{x_0, \ldots, x_{N-1}\}$ and any function, f, with bounded variation, the integration error, ε, is bounded as,

$$\varepsilon[f] \leq V[f] D_N^*. \tag{1}$$

Here $V[f]$ is the total variation of f, in the sense of Hardy-Krause, D_N^*[1] is the star discrepancy of sequence $X = \{x_0, \ldots, x_{N-1}\}$, and $\varepsilon\ [f]$ is defined as

[1]For a sequence of N points $X = \{x_0, \ldots, x_{N-1}\}$ in the d-dimensional unit cube I^d, and for any box, J, with one corner at the origin in I^d, the star discrepancy, D_N^*, is defined as $D_N^* = \sup_{J \in I^d} |\mu_x(J) - \mu(J)|$, where $\mu_X(J) = \frac{\#\text{of points in } J}{N}$ is the discrete measure of J, i. e., the fraction of points of X in J, and $\mu(J)$ is the Lebesgue measure of J, i. e., the volume of J.

$$\varepsilon\ [f] = \int_{I^d} f(x)dx - \frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{2}$$

where d is the dimension of f.

Lemma 1. For M sequences $X_1 = \{x_{0,1}, \ldots, x_{N-1,1}\}$, $X_2 = \{x_{0,2}, \ldots, x_{N-1,2}\}$, …, $X_M = \{x_{0,M}, \ldots, x_{N-1,M}\}$ with discrepancy $D^*_{N,1}, D^*_{N,2}, \ldots, D^*_{N,M}$, respectively, and a function, f, with bounded variation, the integration error ε is bounded as,

$$\varepsilon[f] \le \frac{V[f]}{M}\sum_{i=1}^{M} D^*_{N,i}. \tag{3}$$

Proof: According to the Koksma-Hlawka theorem, the integration error based on the kth quasirandom sequence is

$$\varepsilon_k[f] = \left|\frac{1}{N}\sum_{i=0}^{N-1} f(x_{i,k}) - \int_{I^d} f(x)dx\right| \le V[f]D^*_{N,k}.$$

Then, the sum of these errors is

$$\left|\frac{1}{N}\sum_{k=1}^{M}\sum_{i=0}^{N-1} f(x_{i,k}) - M\int_{I^d} f(x)dx\right| \le \sum_{k=1}^{M}\varepsilon_k[f] =$$

$$\sum_{k=1}^{M}\left|\frac{1}{N}\sum_{i=0}^{N-1} f(x_{i,k}) - \int_{I^d} f(x)dx\right| \le V[f]\sum_{k=1}^{M} D^*_{N,k}.$$

Finally, we can obtain the integration error ε of all these M quasirandom sequences,

$$\varepsilon = \left|\frac{1}{NM}\sum_{k=1}^{M}\sum_{i=0}^{N-1} f(x_{i,k}) - \int_{I^d} f(x)dx\right| \le V[f]\frac{1}{M}\sum_{k=1}^{M} D^*_{N,k} = \frac{V[f]}{M}\sum_{i=1}^{M} D^*_{N,i}.$$

Lemma 1 tells us that the error in the evaluation of an integral based on multiple sequences will be less than or equal to the average of their star discrepancy multiplied by the integral function's total variation. Unfortunately, this upper bound is too coarse. It cannot guarantee that in the *N-out-of-M* scheduling strategy, the evaluation based on the combination of any N out of M quasirandom number sequences, will lead to a smaller error. Figure 2 shows empirical experiments of the *N-out-of-M* strategy in quasi-Monte Carlo integration. In these experiments, we divided a Soboĺ sequence into 4 consecutive blocks, each block having an equal number of quasirandom numbers. Then, we evaluated the integral,

$$\int_0^1 \ldots \int_0^1 x_1x_2x_3x_4dx_1\ldots dx_4,$$

using all combinations of any two of these blocks. In Figure 2, we see that the integral evaluations based on different combinations of the blocks lead to errors that vary considerably. The reason is the combination of two low-discrepancy sequences may not actually yield a smaller discrepancy.

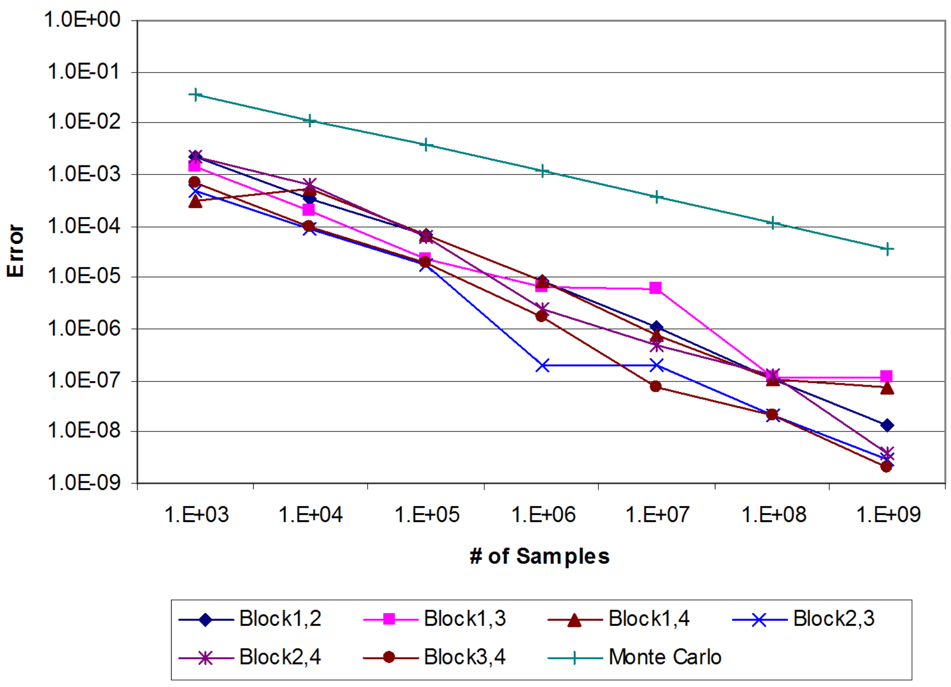

Figure 2: Convergence Analysis of Quasi-Monte Carlo using Different Block Combinations in a Quasirandom Number Sequence.

Even though the computational experiments in Figure 2 show that the combination using blocking or leapfrogging may be effective in the *N-out-of-M* strategies in scheduling quasi-Monte Carlo subtasks, the lack of a theoretical analysis of the discrepancy of the combined sequences limits the use of the *N-out-of-M* strategy in Grid-based quasi-Monte Carlo applications. In comparison to blocking and leapfrog, one can consider taking a single sequence and providing a differently scrambled version of that sequence to the different tasks. Here, each scrambled quasirandom sequence can be thought of as an independent sequence and assigned to a subtask on a Grid node. Under certain circumstances, it can be proven that the scrambled sequence is as uniform as the others [10]. Therefore, the combination of various scrambled quasirandom sequences yields uniformity at the same level as a single sequence with the sum of lengths of all scrambled sequences.

Figure 3 shows the convergence analysis of quasi-Monte Carlo using scrambled Soboĺ sequences on the same integral as in the experiment of Figure 2. These computations are based on a Soboĺ sequence produced by a single generator, a sequence combined with 2 scrambled Soboĺ sequences, and a sequence produced by combining 10 scrambled Soboĺ sequences, respectively. In Figure 3, we see that the computation with a small number of quasirandom numbers has a larger error; however, as the number of random samples grows, the computation based on a single quasirandom sequence and those on sequences of combined scrambled quasirandom sequence tend to a similar convergence rate.

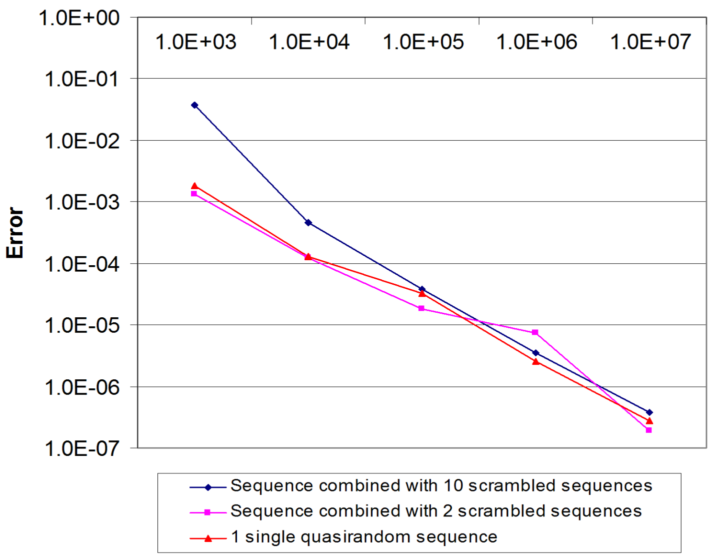

Figure 3: Convergence Analysis of Quasi-Monte Carlo using Scrambled Quasirandom Sequences.

3.2 Lightweight Checkpointing

A subtask running on a node in a Grid system may take a very long time to complete. The *N-out-of-M* strategy is an attempt to mitigate the effect of this on the overall running time. However, if one incorporates checkpointing, he 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 [18]. 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.

To avoid process-level checkpointing, a Monte Carlo computation has a structure highly amenable to application-level checkpointing. Typically, a Monte Carlo application starts in an initial configuration, evaluates a random sample or a random trajectory, estimates a result, accumulates mean and variances with previous results, and repeats this process until some termination condition is met. 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 [4] 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. However, the implementation of application level checkpointing will somewhat increase the complexity of developing new Monte Carlo Grid applications.

Similar to that of pseudorandom numbers, the generation of quasirandom numbers is also deterministic. We can save the status of a quasirandom number generator as well and then reconstruct the quasi-Monte Carlo computation on a Grid node. Therefore, the application-level light-weight checkpointing technique can be simply extended to Grid-based quasi-Monte Carlo applications by storing the status of the quasirandom number generator.

4. Enhancing the Trustworthiness of Grid-Based Quasi-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 `SETI@home` 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.

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, according to 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

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

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

$$s = \left[\frac{1}{n-1}\sum_{i=1}^{n}(f_i - \hat{f})^2\right]^{1/2}.$$

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 we usually have n, the number of subtasks, chosen to be 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. Figure 5 shows the partial result distribution in Grid-based Monte Carlo applications. 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.

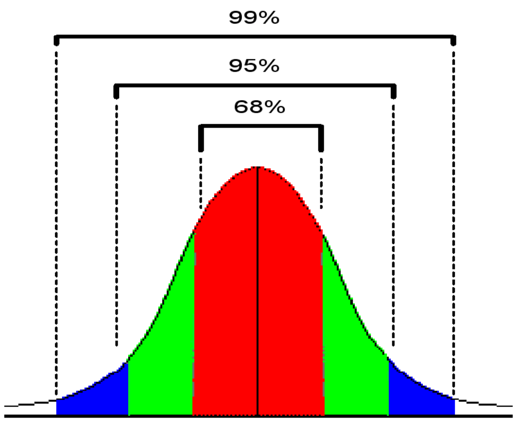

Figure 4: Partial Result Distribution of Grid-based Monte Carlo Applications.

Here is the proposed method for distributed Monte Carlo partial result validation. Suppose we are running n Monte Carlo subtasks on the Grid, the ith subtask will eventually return a partial result, f_i. We anticipate that the f_i's 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 flag events as outliers requires us to choose a multiplier, c, so that we flag events that should only occur 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 suspect. We may either rerun the subtask that generated the suspicious partial result on another node for further validation or just discard it (if using the *N-out-of-M* strategy).

The theoretical foundation of our proposed partial result validation method for Grid-based Monte Carlo applications is the Central Limit Theorem. An important assumption in the Central Limit Theorem is that the underlying random samples are independent. As we know, quasirandom numbers are highly correlated and thus quasi-Monte Carlo computations are deterministic. Hence, quasi-Monte Carlo applications do not share the same statistical nature as Monte Carlo applications. Therefore, the Central Limit Theorem cannot be used here, and so, we cannot expect the partial results from quasi-Monte Carlo subtasks to be normally distributed. The partial validation method based on all partial results in Monte Carlo applications cannot be used in the case of quasi-Monte Carlo.

Nevertheless, here we elucidated an alternative way of partial result validation using a trusted Grid node, which can be easily extended for Grid-based quasi-Monte Carlo applications. To use the validation method, we first need to set up a special subtask that will estimate the same number of samples as the other quasi-Monte Carlo subtasks, but these samples are pseudorandom samples. Secondly, we execute this subtask on a trusted Grid node. Since this subtask is actually a Monte Carlo subtask, we can obtain a confidence interval, $[f_t\text{- } k\sigma_t, f_t + k\sigma_t]$, based on its mean f_t and standard deviation σ_t. Finally, this confidence interval can be used to validate each partial result, f_i, of the quasi-Monte Carlo subtasks running on the potentially untrusted Grid nodes. Figure 5 shows the procedure of the extended partial result validation method for Grid-based quasi-Monte Carlo applications. Due to the fast convergence rate of quasi-Monte Carlo methods, with same number of samples, the quasi-Monte Carlo applications usually have a smaller error than that of the Monte Carlo applications, when the number of samples is big enough. Therefore, we can expect that the partial results of the quasi-Monte Carlo subtasks should also lie in the confidence interval with very high probability. Similar to the partial result validation in Monte Carlo applications, the partial results of the quasi-Monte Carlo subtasks that are not in the confidence interval will be regarded as suspect. The recomputation of such subtasks are recommended.

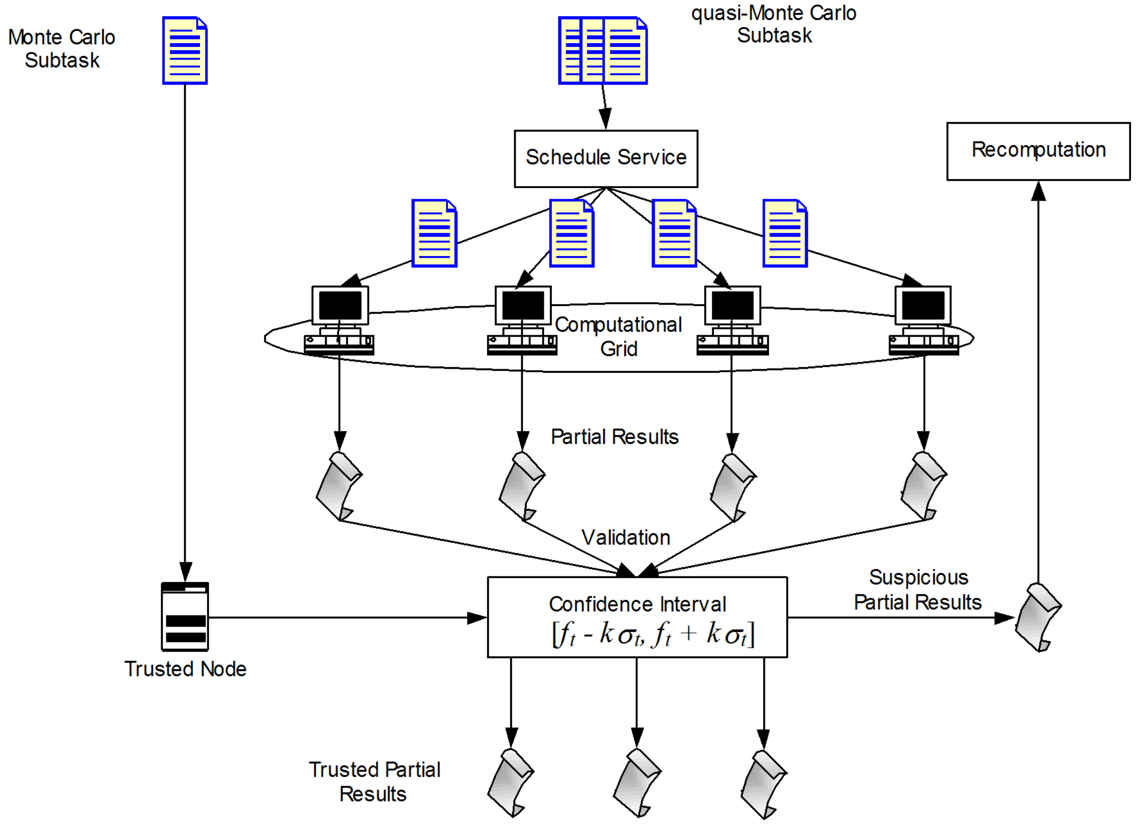

Figure 5: Extended Partial Result Validation Method for Grid-based Quasi-Monte Carlo applications.

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.

The alternative method using a trusted Grid node provides a way to examine the partial results of Grid-based quasi-Monte Carlo applications. However, this method leads to a problem that with quasi-Monte Carlo, the confidence level is not necessarily known. Moreover, the confidence interval generated by the trusted node may be too wide. All these make us turn again to scrambled quasirandom sequences. The basic idea of scrambled quasirandom sequence is to introduce some randomness to the deterministic low-discrepancy sequence so that one can perform error analysis. As a result, confidence intervals can be obtained using the Central Limit Theorem. The confidence interval can then be used to perform partial result validation using a similar approach in Grid-based Monte Carlo applications. Table 1 shows the partial result distribution of 100 Grid-based quasi-Monte Carlo integral evaluation subtasks using different amount of quasirandom numbers from a scrambled Soboĺ sequence. We find that the partial result distribution of quasi-Monte Carlo using scrambled quasirandom number sequence is similar to that of Monte Carlo methods using pseudorandom number sequences. The means of the partial result validation scheme in Grid-based Monte Carlo can then be seamlessly applied to Grid-based quasi-Monte Carlo applications using scrambled quasirandom sequences.

Integrals	N	% $\pm\sigma$	% $\pm 2\sigma$	% $\pm 3\sigma$
$\int_0^1 \dots \int_0^1 x_1 x_2 x_3 x_4 dx_1 \dots dx_4$	10^6	70	95	98
	10^7	77	93	99
$\int_0^1 \dots \int_0^1 \frac{4x_1 x_3^2 e^{2x_1 x_3}}{(1+x_2+x_4)^2} e^{x_5+\dots+x_{20}} x_{21} \dots x_{25} dx_1 \dots dx_{25}$	10^6	68	96	100
	10^7	70	95	99

Table 1: Partial Result Distribution of 100 Grid-based Quasi-Monte Carlo Integral Evaluation Subtasks

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` [24], have the experience that some service providers don't faithfully execute their assigned subtasks. Instead, they attempt to provide bogus partial result at a much lower personal computational cost in order to obtain more benefits per unit cost.

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 some 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 known in advance or are very easy to generate. Therefore, by comparing the intermediate values and these 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, $\ldots$, kNth random numbers used in the subtask. To validate the actual execution of a subtask on the server side, we can just re-compute the Nth, $2N$th, $\ldots$, 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 in 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 decrypted 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.

The intermediate value checking technique takes advantage of the following properties of a pseudorandom number generator: to the Grid node providing computational services, the value of a pseudorandom number remains unknown until it is actually generated; on the other hand, to the application's owner, the value of a pseudorandom number can be easily and economically regenerated or predicted. Leapfrog in quasirandom number sequence can also be implemented. In [7], Bromley illustrates a leapfrog scheme for the Soboĺ sequences, which can actually be used in our intermediate value checking technique for Grid-based quasi-Monte Carlo applications. More importantly, most quasirandom numbers are actually generated not by a recurrence, as with pseudorandom numbers, but with a simple function of i, the quasirandom number in the sequence desired. Thus, it is usually much easier to compute given quasirandom numbers than it is for pseudorandom numbers.

5. Conclusions

Similar to the Monte Carlo applications, quasi-Monte Carlo applications generically also exhibit naturally parallel and computationally intensive characteristics, which can easily fit the dynamic*bag-of-work* model onto a Grid system to implement Grid-based quasi-Monte Carlo computing. Furthermore, using scrambled quasirandom sequences, we extend the techniques used in Grid-based Monte Carlo applications, including an *N-out-of-M* strategy for the efficient scheduling of subtasks on the Grid, lightweight checkpointing for Grid subtask status recovery, a partial result validation scheme to verify the correctness of each individual partial result, and an intermediate result checking scheme to enforce the faithful execution of each subtask, to Grid-based quasi-Monte Carlo applications. These techniques can enhance the performance and trustworthiness of Grid-based quasi-Monte Carlo computing at the application level.

The next phase of our research will be to extend these techniques for Grid-based quasi-Monte Carlo applications to our Grid middleware – the Grid Computing Infrastructure for Monte Carlo Applications (GCIMCA) [15, 16]. At the same time, we will also try to execute more real-life quasi-Monte Carlo applications on our developing Grid system.

References

[1] Condor website. *http://www.cs.wisc.edu/condor.*

[2] Entropia website. *http://www.entropia.com.*

[3] Sprng website. *http://sprng.cs.fsu.edu.*

[4] Xml website. *http://www.xml.org.*

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

[6] Micah Beck, Jack J. Dongarra, Graham E. Fagg, G. Al Geist, Paul Gray, James Kohl, Mauro Migliardi, Keith Moore, Terry Moore, Philip Papadopoulous, Stephen L. Scott, and Vaidy Sunderam. HARNESS: A next generation distributed virtual machine. *Future Generation Computer Systems*, 15(5–6):571–582, 1999.

[7] B. C. Bromley. Quasirandom number generators for parallel Monte Carlo algorithms. *Journal of Parallel and Distributed Computing*, 38(1):101–104, 1996.

[8] Rajkumar Buyya, Steve J. Chapin, and David C. DiNucci. Architectural models for resource management in the grid. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 18–35. Springer-Verlag, 2000.

[9] R. E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta Numerica*, 8:1–49, 1998.

[10] H. Chi and M. Mascagni. Scrambled quasirandom sequences and their application. submitted for publication in SIAM Review, 2003.

[11] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[12] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150:1–17, 2001.

[13] Hee Sun Hong and Fred J. Hickernell. Algorithm 823: Implementing scrambled digital sequences. *ACM Transactions on Mathematical Software*, 29(2):95–109, June 2003.

[14] Yaohang Li and Michael Mascagni. Grid-based Monte Carlo application. *Lecture Notes in Computer Science*, 2536:13–24, 2002.

[15] Yaohang Li, Michael Mascagni, and Robert van Engelen. Gcimca: A globus and sprng implementation of a grid computing infrastructure for monte carlo applications. In *Proceeding of the International Multiconference in Computer Science and Computer Engineering, PDPTA'03.* IEEE, 2003.

[16] Yaohang Li, Michael Mascagni, Robert van Engelen, and Q. Cai. A grid workflow-based monte carlo simulation environment. submitted for Journal of Neural Parallel and Scientific Computations, 2003.

[17] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[18] Miron Livny, Jim Basney, Rajesh Raman, and Todd Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.

[19] Michael Mascagni and Ashok Srinivasan. Algorithm 806: SPRNG: a scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software*, 26(3):436–461, September 2000.

[20] Michael O. Neary, Bernd O. Christiansen, Peter Cappello, and Klaus E. Schauser. Javelin: Parallel computing on the Internet. *Future Generation Computer Systems*, 15(5–6):659–674, 1999.

[21] Luis F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Gener. Comput. Syst.*, 18(4):561–572, 2002.

[22] A. Srinivasan. Parallel and distributed computing issues in pricing financial derivatives through quasi-monte carlo. In *Proceedings of the Sixteenth International Parallel and Distributed Processing Symposium*. IEEE, 2002.

[23] A. Srinivasan, D. M. Ceperley, and M. Mascagni. Random number generators for parallel applications. *Monte Carlo Methods in Chemical Physics*, 105:13–36, 1998.

[24] Dan Werthimer, Jeff Cobb, Matt Lebofsky, David Anderson, and Eric Korpela. Seti@home: massively distributed computing for seti. *Comput. Sci. Eng.*, 3(1):78–83, 2001.