GCIMCA: A Globus and SPRNG Implementation of a Grid-Computing Infrastructure for Monte Carlo Applications

Yaohang Li, Michael Mascagni, Robert van Engelen
Department of Computer Science and School of Computational Science and Information Technology
Florida State University
Tallahassee, FL 32306-4530, USA
{yaohanli, mascagni, engelen}@cs.fsu.edu

Abstract

The implementation of large-scale Monte Carlo computation on the grid benefits from state-of-the-art approaches to accessing a computational grid and requires scalable parallel random number generators with good quality. The Globus software toolkit facilitates the creation and utilization of a computational grid for large distributed computational jobs. The Scalable Parallel Random Number Generators (SPRNG) library is designed to generate practically infinite number of random number streams with favorable statistical properties for parallel and distributed Monte Carlo applications. Taking advantage of the facilities of the Globus toolkit and the SPRNG library, we implemented a tool we refer to as the Grid-Computing Infrastructure for Monte Carlo Applications (GCIMCA). GCIMCA implements services specific to grid-based Monte Carlo applications, including the Monte Carlo subtask schedule service using the N-out-of-M strategy, the facilities of application-level checkpointing, the partial result validation service, and the intermediate value validation service. Based on these facilities, GCIMCA intends to provide a trustworthy grid-computing infrastructure for large-scale and high-performance distributed Monte Carlo computations.

1. Introduction

Monte Carlo applications are widely perceived as computationally intensive but naturally parallel. With more ambitious calculations by estimating more random samples, a Monte Carlo application is capable of reducing the statistical errors to any desired level [1]. By computing and analyzing random samples independently, Monte Carlo applications can be programmed in a bag-of-work model and fit into the master-worker paradigm. In a parallel environment using the master-worker paradigm, the master partitions the task, schedules subtasks to workers, and receives results when the workers complete their assigned work [2]. The subsequent growth of computer power, especially that of the parallel and distributed computing systems, has made large-scale distributed Monte Carlo computation possible and practically effective.

Large-scale Monte Carlo computation consumes large amounts of computational power, and depends on parallel random number generators with good quality. On the one hand, 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 [3]. A computational grid based on the grid-computing techniques can, in principle, provide a tremendously large amount of CPU cycles to a Monte Carlo application. The Globus software toolkit [4] provides software tools and services to build computational grid infrastructures for grid-based applications. On the other hand, the SPRNG (Scalable Parallel Random Number Generators) [5] library is designed to use parameterized pseudorandom number generators to provide independent random number streams. The SPRNG library provides uniform programming interfaces for the Linear Congruential Generator (LCG), Prime Modulus Linear Congruential Generator (PMLCG), additive Lagged-Fibonacci Generator (LFG), Multiplicative Lagged-Fibonacci Generator (MLFG), and Combined Multiple Recursive Generator (CMRG). Some generators in the SPRNG library can provide up to $2^{78000} - 1$ independent random number streams [6] with sufficiently long period, which have favorable inter-stream and cross-stream properties in a statistical sense. These generators can meet the random number requirements of most distributed Monte Carlo applications. Furthermore, by analyzing the statistical nature of Monte Carlo applications and the cryptographic aspects of these underlying random number generators, our previous research [7, 8, 9] developed techniques to improve the performance and trustworthiness of Monte Carlo computations on the grid. For large-scale grid-based Monte Carlo analysis, "all the pieces of the puzzle" have recently been assembled. In this paper, we are going to elucidate the implementation of the Grid-Computing Infrastructure for Monte Carlo Applications (GCIMCA) utilizing the Globus toolkit and SPRNG. Also, we will discuss the services provided by GCIMCA based on the fortuitous characteristics of Monte Carlo applications.

The remainder of this paper is organized as follows. We illustrate the architecture and the working paradigm of GCIMCA in Sections 2 and 3, respectively. In Section 4, we discuss detailed implementations of the core services and facilities in GCIMCA. Finally, Section 5 summarizes our conclusions and future research directions.

2. Architecture of GCIMCA

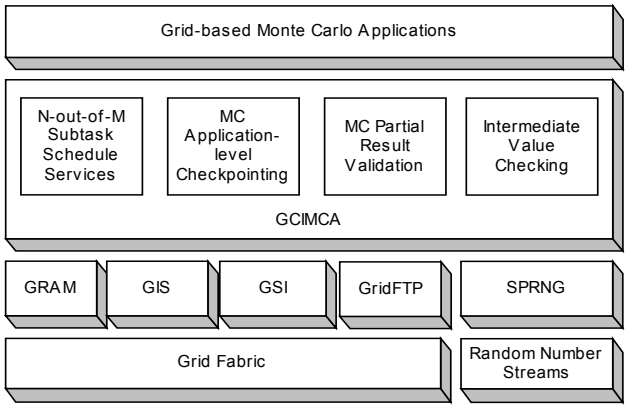

Figure 2.1 Architecture of GCIMCA

GCIMCA is designed on the top of the grid services provided by Globus, [10] and supplies facilities and services for grid-based Monte Carlo applications. The services include GRAM (Globus Resource Allocation Manager), GIS (Grid Information Service), GSI (Grid Security Infrastructure), and GridFTP. GRAM is used to do Monte Carlo subtask remote-submission and manage the execution of each subtask. GIS provides information services, i.e., the discovery of the properties and configurations of grid nodes. GSI offers security services such as authentication, encryption and decryption for running Monte Carlo applications on the grid. GridFTP provides a uniform interface for data transport and access on the grid for GCIMCA. At the same time, the execution of each Monte Carlo subtask usually consumes a large amount of random numbers. SPRNG is the underlying pseudorandom number generator library in GCIMCA, providing independent pseudorandom number streams. Based on the grid services provided by Globus and the SPRNG library, GCIMCA provides higher-level services to grid-based Monte Carlo applications. These services include *N-out-of-M* Monte Carlo subtask scheduling, application-level checkpointing, partial result validation, and intermediate value checking. Figure 2.1 shows the architecture of GCIMCA.

3. GCIMCA Working Paradigm

3.1 Overview

Figure 3.1 shows the GCIMCA working paradigm, which is based on the master-worker model of a grid-based Monte Carlo application. The execution of a grid-based Monte Carlo application in GCIMCA is initiated by a user who submits a Monte Carlo job description to the job server. At the same time, the user prepares and stores the Monte Carlo job files, such as the executable binary and data files, on a job file server. The job file server may run in the user's own domain and allow only those accesses with authentication. The GCIMCA job server manages the subtasks of a Monte Carlo job, and is in charge of actually scheduling these subtasks. The eligible GCIMCA subtask agent running on a node within an organization obtains a Monte Carlo subtask description from the job server. Then, according to the specification of the subtask description, the subtask agent downloads the necessary files from the job file server using GridFTP and retrieves the actual subtask. The subtask is set remotely running on an eligible grid node within the organization by GRAM. When a subtask is ready, the partial result files are submitted back to the job file server. At the same time, the job server is notified that a subtask is done. When the entire Monte Carlo job is finished, the partial results are validated and the job server notifies the user as to the completion of the computation.

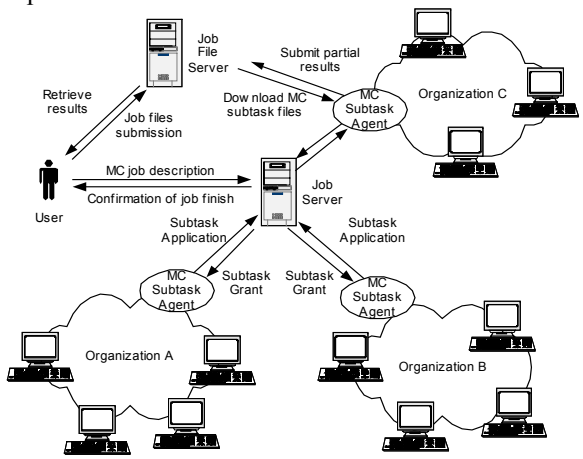

Figure 3.1 The Working Paradigm for GCIMCA

3.2 Job Submission

Monte Carlo Job Description

```
JobName =                "Monte Carlo Integration"
JobDescription = "Execfile=http://sprng.cs.fsu.edu/mcint/mcintIntel.out
                         Datafile=http://sprng.cs.fsu.edu/mcint/mcint.data
                         Arg= -r
                         Arch=INTEL Opsys=LINUX"
JobDescription= "Execfile=http://sprng.cs.fsu.edu/mcint/mcintSolaris.out
                         Datafile=http://sprng.cs.fsu.edu/mcint/mcint.data
                         Arg= -r
                         Arch=SUN  Opsys=Solaris26"
RequiredJobs =           20
MaxJobs =                40
ResultFileName =         mcintresult.dat
ResultLocation =         http://sprng.cs.fsu.edu/mcint/result
Org=                     cs.fsu.edu;csit.fsu.edu
Encryption= YES
```

Figure 3.2 Sample of a Monte Carlo Job Description File

A user submits a Monte Carlo job description file to the job server. The Monte Carlo job description file declares the information related to the Monte Carlo job, including the job name, locations of executable and data files, arguments, required hardware architectures and operating systems, number of subtasks, result file names

and destinations, encryption option, and authenticated organization. Figure 3.2 shows a sample of a job description file. Based on the job description, the job server validates the Monte Carlo job, creates a Monte Carlo subtasks pool, chooses the qualified subtask applications from the subtask agent, verifies the authentication of a subtask agent using GSI, and then actually schedules the subtask.

In GCIMCA, a user provides different executable binary files for each possible different system architecture on the grid. The remote compiler [15] service is used to address this heterogeneity issue. A user can send source packages to a remote node of a specific system architecture with the remote compiler service running. Then, the remote compiler service compiles the source files, generates the executable files, and sends them back to the user. Using the remote compiler service, different executable codes for different platforms can be obtained.

3.3 Passive-Mode Subtask Scheduling

Unlike the design of most existing distributed and parallel computing systems, such as Condor [11], Javelin [12], Charlotte [13] and HARNESS [14], which use an active scheduling mode to dispatch subtasks, GCIMCA uses a passive scheduling mode. In an active scheduling mode, the job server needs to keep checking the status of computational nodes to schedule tasks to the capable ones. Also, the job server must keep track of each running subtask. In contrast, using the passive scheduling mode in GCIMCA, a Monte Carlo subtask agent sends applications to the job server to apply for a subtask only when it has computational nodes available and ready for work. The management responsibility of the execution of each subtask is decentralized to the subtask agents. The advantage of using the passive scheduling mode here is to reduce the workload, or more specifically, the requirements of network connection bandwidth of the job server. In GCIMCA, most of the communication load is between a subtask agent and the computational nodes within the organization usually having connection via a high-speed LAN. The communication between the job server and the subtask agents, which is usually through a WAN with relatively low bandwidth, is minimized.

In GCIMCA, the job server manages the jobs from the users, and processes subtask applications from the subtask agents. It is the subtask agent that retrieves the information related to a subtask, forms the subtask described in Globus RSL (Resource Specification Language), and actually schedules the subtask to a grid node. The job management functionalities of GRAM are utilized to run subtasks on a remote grid node. Figure 3.3 shows the GCIMCA implementation of remotely executing a Monte Carlo subtask based on GRAM. When a Monte Carlo subtask is scheduled on a grid node, a process running the GCIMCA subtask callback function is created so as to listen to the status as it changes on the running subtask. Depending on the status of the running subtask, the callback function takes corresponding actions, such as reporting to the job server, submitting partial result files, or rescheduling the subtask with checkpoint data.

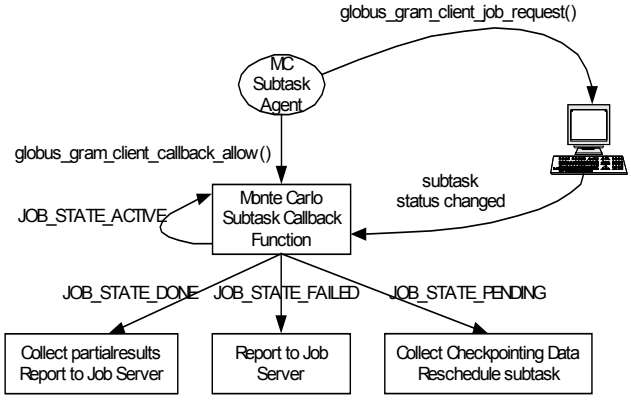

Figure 3.3 Remote Execution of a Monte Carlo Subtask

4. Implementation of GCIMCA Services

4.1 *N-out-of-M* Scheduling Strategy

The main idea of the *N-out-of-M* strategy [7, 8, 9] for grid-based Monte Carlo computations is to schedule more subtasks than are required to tolerate possible delayed or halted subtasks on the grid to achieve optimal performance. 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 stream, and we submit M instead of N subtasks to the grid system. When N partial results are ready, we consider the whole task for the grid system to be completed. More theoretical analysis of the *N-out-of-M* strategy can be found in [8, 9].

Figure 4.1 shows the implementation of the *N-out-of-M* scheduling strategy in GCIMCA. The Monte Carlo job description file from the user states the maximum number (M) of subtasks to be scheduled and the required number (N) of those to achieve a certain predetermined accuracy. Based on this, the GCIMCA job server sets up a subtask pool with the number of entries as M. Each entry of the pool describes the status of a subtask, including the subtask schedule status, random stream ID for the SPRNG library, the responsible subtask agent if scheduled, and other implementation dependent details. The job server also maintains the statistics of completed subtasks. Once the number of completed subtasks reaches the number of requested subtasks, the job server will regard this Monte Carlo job as complete. A subtask-canceling signal will be sent to the subtask agents that still have subtasks running related to this job.

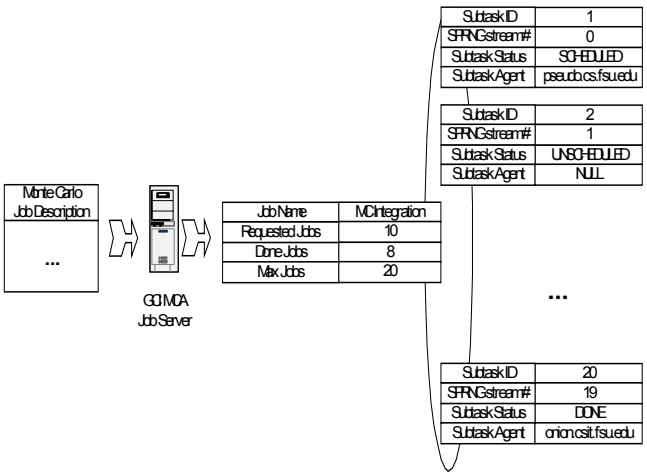

Figure 4.1 Implementation of the *N-out-of-M* Scheduling Strategy in GCIMCA

4.2 Monte Carlo Lightweight Checkpointing

A long-running computational task on a grid node must be prepared for node unavailability. Compared to process-level checkpointing [11], application-level checkpointing is much smaller in size and thus less costly. More importantly, the application-level checkpointing data is usually readily portable and is easy to migrate from one platform to another. Monte Carlo applications have a structure highly amenable to application-level checkpointing. Typically, a Monte Carlo application can be programmed in a structure that 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 conditions are met.

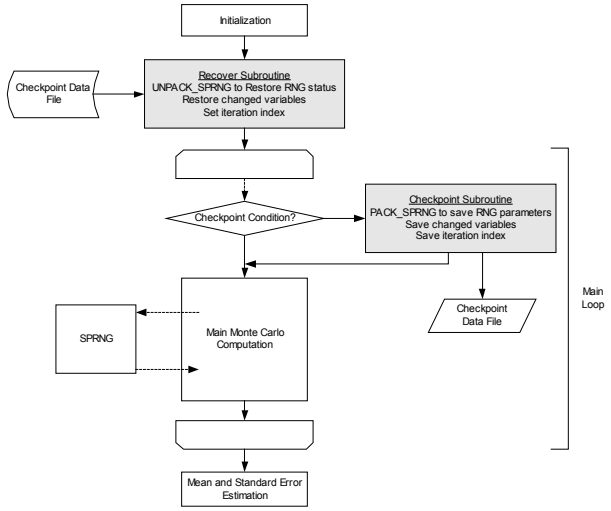

Figure 4.2 GCIMCA Implementation of Monte Carlo Application-Level Checkpointing

Thus, to recover an interrupted computation, a Monte Carlo subtask needs to save only a relatively small amount of information, which includes 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. GCIMCA uses the pack_sprng() and unpack_sprng() functions [6] in the SPRNG library to store and recover the states of random number streams, respectively. At the same time, GCIMCA requires the Monte Carlo application programmer to specify the other checkpoint data, and also the location of the main loop to generate the checkpointing and recovery subroutines. Figure 4.2 shows the flowchart of GCIMCA's implementation of Monte Carlo application-level checkpointing and recovery.

4.3 Partial Result Validation and Intermediate Value Checking

Grid-based Monte Carlo applications are very sensitive to each partial result generated from subtasks running on the widely distributed grid nodes. An erroneous computation of a subtask will most likely lead to the corruption of the whole grid Monte Carlo computation. To enforce the correctness and accuracy of grid-based Monte Carlo computations, GCIMCA provides a partial result validation service and an intermediate value checking service.

The partial result validation service takes advantage of the statistical nature of distributed Monte Carlo applications. In distributed Monte Carlo applications, we anticipate the partial results are approximately normally distributed. Based on all the partial results and a desired confidence level, the normal confidence interval is evaluated. Then, each partial result is examined. If it is in the normal confidence interval, this partial result is considered as trustworthy; otherwise it is very suspicious. Discussion of the grid-based Monte Carlo partial result validation can be found in [7, 8]. To utilize the partial result validation service, GCIMCA requires the user to specify the quantities in the partial result data files that are anticipated to conform to the approximately normal distribution. Then, when the Monte Carlo job is done, the job server will collect all these value files from the subtask agents using GridFTP, compute the normal confidence interval, and begin to check each partial result. If a partial result is found suspicious, the job server will reschedule the particular subtask that produced this partial result on another grid node to perform further validation.

The intermediate value checking service is used to check if the assigned subtask from a grid node is faithfully carried out and accurately executed. The 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 in the program. On the other hand, to the owner of the application, certain intermediate values are either pre-known or very easy to generate. By comparing the intermediate values and the pre-known values, we can control whether the subtask is actually faithfully executed. The underlying pseudorandom numbers in the Monte Carlo applications are the perfect candidates to use as the intermediate values [8]. The intermediate value checking service in GCIMCA uses a simple strategy to validate a result from subtasks by tracing certain predetermined random numbers in the grid-based Monte Carlo applications. To utilize the intermediate value checking service, GCIMCA also requires user-level cooperation. The application programmers need to save the value of the current pseudorandom number after every *N* pseudorandom numbers are generated. Thus, a record of the *N*th, *2N*th, …, *kN*th random numbers used in the subtask are produced. When a subtask is done, the GCIMCA job server obtains this record and then re-computes the *N*th, *2N*th, …, *kN*th random numbers applying the specific generator in the SPRNG library with the same seed and parameters as used in this subtask. A mismatch indicates problems during the execution of the subtask.

5. Conclusions

Monte Carlo applications generically exhibit naturally parallel and computationally intensive characteristics. In this paper, we discussed utilizing the Globus software toolkit and the SPRNG library to implement GCIMCA -- a grid-computing infrastructure for Monte Carlo applications. The *N-out-of-M* subtask schedule service, application-level checkpointing service, Monte Carlo partial result validation service, and intermediate value checking service are implemented in GCIMCA to provide grid-computing facilities for trustworthy and high-performance large-scale Monte Carlo computation purposes.

In the future, we plan to adopt the emerging OGSA (Open Grid Services Architecture) [16] into GCIMCA so that we can integrate the standard grid-computing services into grid-based Monte Carlo applications. Also, we plan to implement the remote checkpointing facilities using gSOAP [17] for Monte Carlo application-level checkpointing in a grid-computing environment. At the same time, we will also try to apply more real-life Monte Carlo applications on GCIMCA. A final goal is to experiment with the Monte Carlo based services on non-Monte Carlo applications. The goal here being to study the extent to which these application-specific services can enhance other grid computations.

References

[1] A. Srinivasan, D. M. Ceperley, M. Mascagni, "Random Number Generators for Parallel Applications," Monte Carlo Methods in Chemical Physics, **105**:13-36, 1999.

[2] J. Basney, R. Raman, M. Livny, "High Throughput Monte Carlo," Proc. of 9th SIAM Conf. on Parallel Processing for Sci. Comp., San Antonio, 1999.

[3] I. Foster, C. Kesselman, S. Tueske, "The Anatomy of the Grid," Intl. Jour. of Supercomp. App., **15**(3), 2001.

[4] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," Intl. Jour. of Supercomputer App., **11**(2):115-128, 1997.

[5] M. Mascagni, A. Srinivasan, "SPRNG: A Scalable Library for Pseudorandom Number Generation," ACM Transactions on Mathematical Software, 2000.

[6] SPRNG website, http://sprng.cs.fsu.edu.

[7] Y. Li, M. Mascagni, "Grid-based Monte Carlo Application," Lecture Notes in Computer Science, **2536**:13-25, GRID2002, Baltimore, 2002.

[8] Y. Li, M. Mascagni, "Analysis of Large-scale Grid-based Monte Carlo Applications," special issue of Intl. Jour. of High Performance Comp. App., 2003.

[9] Y. Li, M. Mascagni, "Improving Performance via Computational Replication on a Large-Scale Computational Grid," IEEE/ACM CCGRID2003, Tokyo, 2003.

[10] Globus website, http://www.globus.org.

[11] M. Litzkow, M. Livny, M. Mutka, "Condor - A Hunter of Idle Workstations," Proc. of 8th Intl. Conf. of Dist. Comp. Systems, pp. 104-111, 1988.

[12] B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, D. Wu, "Javelin: Internet-Based Parallel Computing Using Java," Concurrency: Practice and Experience, **9**(11): 1139 - 1160, 1997.

[13] A. Baratloo, M. Karaul, Z. Kedem, P. Wyckoff, "Charlotte: Metacomputing on the Web," 9th Intl. Conf. on Parallel and Dist. Comp. Systems, 1996.

[14] Beck, Dong, Fagg, Geist, Gray, Kohl, Miliardi, K. Moore, T. Moore, P. Papadopoulous, S. Scott, V. Sunderam, "HARNESS: a next generation distributed virtual machine," Jour. of Future Generation Computer Systems, (15), 1999.

[15] M. Zhou, "A Scientific Computing Tool for Parallel Monte Carlo in a Distributed Environment," Ph.D. Dissertation, Univ. of Southern Mississippi, 2000.

[16] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, "The Physiology of Grid: Open Grid Services Architecture for Distributed Systems Integration," draft, 2003.

[17] R. van Engelen, K. A. Gallivan, "The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks," Proc. of ACM/IEEE CCGrid02, Berlin, 2002.