THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

A GRID COMPUTING INFRASTRUCTURE FOR MONTE CARLO APPLICATIONS

BY

YAOHANG LI

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Summer Semester, 2003

The members of the Committee approve the dissertation of
Yaohang Li defended on May 5th, 2003.

Michael Mascagni
Professor Directing
Dissertation

Michael H. Peters
Outside Committee Member

Craig Nolder
Outside Committee Member

David Whalley
Committee Member

Robert van Engelen
Committee Member

Xin Yuan
Committee Member

Sudhir Aggarwal, Chair, Department of Computer Science

The Office of Graduate Studies had verified and approved the above named committee members.

To my father and mother.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere thanks and appreciation to my advisor and "cheer leader," Dr. Michael Mascagni for his continuous encouragement and kind help in my Ph.D. research and study. Also, I give thanks to all my other members in my committee, Dr. David Whalley, Dr. Robert van Engelen, Dr. Xin Yuan, Dr. Michael H. Peters, and Dr. Craig Nolder, for their helpful comments and advice. They have added another dimension to my research.

I wish to acknowledge the CSIT fellowship and computational resources provided by the School of Computational Science and Information Technology at the Florida State University and dissertation research grant by the Graduate Study Office at the Florida State University. At the same time, I appreciate the help from our research group, especially Dr. Aneta Karaivanova, Dr. Chi-Ok Hwang, Dr. Nikolai Simonov, Dr. Abdujabor Rasulov, Hongmei Chi, Jason Parker, and Wenchang Yan's creative discussion.

I express my deepest gratitude and respect to my parents in China. Also, in heart, I will forever remember all of my friends who supported me and helped me, especially during my difficult times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Monte Carlo applications are widely perceived as computationally intensive but naturally parallel. Therefore, they can be effectively executed on the grid using the dynamic bag-of-work model. We improve the efficiency of the subtask-scheduling scheme by using an *N-out-of-M* strategy, and develop a Monte Carlo-specific lightweight checkpoint technique, which leads to a performance improvement for Monte Carlo grid computing. Also, we enhance the trustworthiness of Monte Carlo grid-computing applications 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 our implementation of a grid-computing infrastructure – GCIMCA (Grid-Computing Infrastructure for Monte Carlo applications), which is based on Globus and the SPRNG (Scalable Parallel Random Number Generators) library. GCIMCA intends to provide trustworthy grid-computing services for large-scale and high-performance distributed Monte Carlo computations.

We apply Monte Carlo applications to GCIMCA to show the capability of our techniques. These applications include the grid-based Monte Carlo integration and a "real-life" Monte Carlo application -- the grid-based hybrid Molecular Dynamics (MD)/Brownian Dynamics (BD) application for simulating the long-time, nonequilibrium dynamics of receptor-ligand interactions. Our preliminary results show that our techniques and infrastructure can achieve significant speedup, efficiency, accuracy, and trustworthiness for grid-based Monte Carlo applications.

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

Recently, grid computing has emerged as an important new area in parallel computing, distinguished from traditional distributed computing by its focus on large-scale dynamic, distributed, and heterogeneous resource sharing, cooperation of organizations, innovative applications, and high-performance orientation. Many applications have been developed to take advantage of grid computing facilities and have already achieved elementary success. However, as the field grew so also did the problems associated with it. Traditional assumptions that are more or less valid in traditional distributed and parallel computing settings break down on the Grid. In traditional distributed computing settings, one often assumes a "well-behaved" system: no faults or failures, minimal security requirements, consistency of state among application components, availability of global information, and simple resource sharing policies. While these assumptions are arguably valid in tightly coupled systems, they break down as systems become much more widely distributed [1]. Therefore, despite the undisputed success of grid computing, as seen in SETI@home [2, 3] and distributed.net [4], many fundamental problems and questions remain unanswered, such as, system performance, heterogeneous resources management, task-schedule efficiency, security, portability, reliability, and trustworthiness of computing.

Among all these problems, from the grid application point-of-view, two of them are of prime importance. One is the performance, i.e., how to efficiently manage and use the large-scale, widely distributed resources in the grid to reduce the application task completion time. The other is the trustworthiness, i.e., how to guarantee that the computational results obtained from the grid are due to the grid application requested [5]. Like most of the fundamental issues in computer science, these problems are addressed at the system level, programming level, or application level. At the system level, there have been efforts focused on developing services, functionalities, and protocols to satisfy the requirements of grid applications. At the programming level, Application Programming Interfaces (APIs) and Software Development Kits (SDKs) have been constructed to provide the programming abstractions required to create a usable and reliable grid. At the application level, proper applications are picked up, the characteristics of the application are examined, and then approaches are developed to address their issues for grid computing. The goal in which all these efforts at different levels converge is to achieve high-performance and trustworthy grid computation.

Monte Carlo methods comprise a branch of experimental mathematics that is concerned with experiments using random numbers. They are important techniques in performing simulations, optimization and integration, and have been employed sporadically in numerous fields of science and engineering, including nuclear physics [6, 7], medicine [8], chemistry [9], meteorology [10], and biology [11]. At the same time, applications employing Monte Carlo methods are widely perceived as computationally intensive but naturally parallel. Therefore, Monte Carlo applications are considered as a natural fit for grid computing. They are expected to be effectively executed using the dynamic *bag-of-work* model [12], which splits a big computational task into smaller independent subtasks. Programmed via the dynamic *bag-of-work* model, large-scale Monte Carlo computations can then be deployed on the grid. At the same time, Monte Carlo applications and the underlying random number generators exhibit interesting properties [13], which may be used to address important issues of grid computing at the

application level. In a word, the inherent characteristics of Monte Carlo applications motivate the use of the grid techniques to effectively perform large-scale Monte Carlo computation.

Just like many other grid applications, large-scale grid-based Monte Carlo applications have to confront the challenges of the existing issues in grid computing. Further research is needed to tackle the problems of performance and trustworthiness of applying grid-computing techniques to Monte Carlo applications. The aim of this dissertation research is to take advantage of the characteristics of Monte Carlo applications to develop application-level approaches and establish a grid-computing infrastructure based on these approaches for high-performance and reliable large-scale grid-based Monte Carlo computations.

1.2 Terminology

Traditionally, a parallel computing system is defined as one composed of tightly coupled processors that can coordinate to accomplish the concurrent solution of a common task [14]. In a parallel computing system, the processors typically work in tight synchrony, share memory to a large extent, and have very fast and reliable communication channels between them. However, in this dissertation research paper, we will instead focus on distributed computing systems where the processors are thought of as being more loosely coupled, which is a result of less rapid intercommunication and inherently larger grained processing performed on the different processors, computers, or workstations. Distributed computing systems typically present different problems than those of parallel systems, since each of the processors is autonomous and may refuse a request for services, and the processors here are connected by a heterogeneous network. Technically, the meaning of meta-computing is very close to that of distributed computing, with an emphasis on problems arising from the extreme heterogeneity of such systems. At the same time, grid computing has emerged as an important new field, defined as “a single seamless computational environment in which cycles,

communication, and data are shared" [15], denoting the construction of a distributed computing infrastructure focusing on large-scale resource sharing [16]. According to their functionalities, grids can be classified as a computational grid, an access grid, a data grid, and a data-centric grid [17]. In this dissertation, we are especially interested in the computational grid, which has emphasis on high-performance computing and large-scale data sharing.

1.3 Dissertation Objectives

The purpose of this research is to provide guidance and develop techniques for large-scale grid-based Monte Carlo computations. Our approach is to investigate the statistical nature of Monte Carlo applications and the cryptographic aspect of the underlying random number generators to develop application-level techniques to address performance and trustworthiness issues in grid-based Monte Carlo computation. The objectives of this dissertation are:

1) to research and analyze the inherent characteristics of grid-based Monte Carlo applications;
2) to develop approaches and techniques to address the performance and trustworthiness issues of Monte Carlo computation on a computational grid from the application level;
3) to build up a grid-computing infrastructure for high-performance and trustworthy large-scale Monte Carlo computations based on these techniques; and
4) to extend these techniques to other applications on the grid.

1.4 Dissertation Organization

The remainder of this dissertation paper is organized as following. We present a literature review of grid computing and Monte Carlo method in Chapter 2. In Chapter 3, we analyze the characteristics of distributed Monte Carlo applications and their

underlying random number generators. The analysis leads to techniques to improve performance and trustworthiness of grid-based Monte Carlo applications. Based on these techniques, in Chapter 4, we elucidate our implementation of a grid-computing middleware tool, which we refer to as the Grid-Computing Infrastructure for Monte Carlo Applications (GCIMCA), based on the Globus software toolkit and the SPRNG library. We apply the Monte Carlo applications to GCIMCA and we present the preliminary computational results in Chapter 5. These applications include the grid-based Monte Carlo integration and the grid-based hybrid Molecular Dynamics (MD)/Brownian Dynamics simulation. In Chapter 6, we discuss the extension of techniques in GCIMCA for other applications. Finally, Chapter 7 summarizes our conclusions and provides our future (post-dissertation) research directions.

CHAPTER 2

LITERATURE REVIEW

In this literature review, we will provide a review of a limited number of studies related to distributed computing and the grid. Going through the history of the grid, studying the approaches in active grid projects, and analyzing the working paradigm of a computational grid in Section 2.1, this review seeks to present what the previous research has discovered about these problems. Following the review of grid computing, in Section 2.2, we will also present a survey of distributed Monte Carlo applications. In this Monte Carlo application survey, we will study previous work exploring the power of distributed computing for large-scale Monte Carlo applications. In a word, the literature review shows the link between grid computing and distributed Monte Carlo applications and reveals existing issues and problems, which forms the foundation of our dissertation research.

2.1 Grid Computing

2.1.1 From Distributed Computing to Grid Computing

In the early 1970s, when computers were first linked by networks, the idea of harnessing distributed computing power was born. A few early experiments with distributed computing, including a pair of programs called Creeper and Reaper, ran on the Internet's predecessor, the ARPAnet. In 1973, the Xerox Palo Alto Research Center installed the first Ethernet network and the first full-fledged distributed computing effort

was underway. Scientists J. F. Shoch and J. A. Hupp created a program called "worm," which routinely cruised about 100 Ethernet-connected computers. These worms could move from machine to machine using idle resources for beneficial purposes. Each worm used idle resources to perform a computation and had the ability to reproduce and transmit clones to other nodes of the network. With these worms, Shoch and Hupp distributed graphic images and shared computations for rendering realistic computer graphics. In another effort, R. Crandall started putting idle, networked NeXT computers to work. Crandall installed software that allowed the machines, when not in use, to perform computations and to combine efforts with other machines on the network. His software, named "Zilla," first focused on finding, factoring, and testing the primality of huge numbers, and then moved on to test encryption [18].

Subsequent exploration of distributed computing centered on using a network-connected cluster. In 1988, M. Livny at the University of Wisconsin, Madison, created Condor, a software system that put the University's idle computers to work to provide High Throughput Computing (HTC) [19, 20, 21]. Also in 1988, A. Lenstra and M. Manesse of the DEC Systems Research Center wrote a software program that distributed factoring tasks to computers inside and outside their Palo Alto, California lab via email [22].

Distributed computing scaled to a global level with the maturation of the Internet in the 1990s. In 1996, the Great Internet Mersenne Prime Search (GIMPS) [23] project used distributed computing to search for enormous Mersenne prime numbers. C. Percival launched PiHex [24], a successful distributed computing effort to calculate the digits of π. Among the many new distributed computing projects, two projects in particular have proven that the concept works extremely well, even better than many experts had anticipated.

- The first of these revolutionary projects is distributed.net, which used thousands of independently owned computers across the Internet to crack encryption codes.

- The second, and the most successful and popular of distributed computing projects in history, is the SETI@home project [2, 3]. Over two million people, the largest number of volunteers for any Internet distributed computing project to date, have installed the SETI@home software agent since the project started in May 1999. This project conclusively proved that distributed computing could accelerate computing project results while managing project costs.

Later, more and more projects in many different areas utilized and demonstrated the power of distributed computing. In 2000, Stanford scientists launched Folding@home [25] for protein folding simulation. NASA launched clickworkers [26], a project to search for craters on Mars. Intel-United Device's Cancer Research Project [27] was launched for searching drugs for use in cancer therapy. All these projects use the Internet distributed computing paradigm.

As more and more applications use the globally distributed computing technique, many problems and challenges arise as well. First of all, how to efficiently manage and utilize the vast and widely distributed dynamic resources becomes an unavoidable task to every globally distributed computing project. Secondly, the distributed computations must worry about the trustworthiness of these computations performed on a likely "untrustable" computer. For example, experience with SETI@home has shown that users may fake computations and return wrong or inaccurate results to obtain more rewarded benefits [28]. Thirdly, computations in a distributed computing environment have to face the problems that arise from the possible sudden unavailability of certain computers or segments of the network. Thus, performance, reliability, trustworthiness, scalability, and security become critical issues in distributed computing applications. In large-scale distributed computing environments, these problems become more abrupt.

To address these issues, many of the more recent efforts in distributed computing are aimed at developing a general-purpose distributed computing infrastructure,

providing integrated security, availability, scalability, reliability, and manageability for general distributed computing applications. In the mid 1990s, the term "grid" was coined by I. Foster and C. Kesselman in their book, "The Grid: Blueprint for a New Computing Infrastructure," to denote a proposed distributed computing infrastructure for advanced science and engineering [29]. In a very short period of time, grid computing was widely accepted in popular perception, denoting a framework for "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institution and resources" [30].

2.1.2 Grid-Computing Projects

In addition to application-oriented projects like SETI@home, Folding@home, and distributed.net, there are many academic and industrial projects that aim at developing an infrastructure or framework for general grid computing applications. For instance, Charlotte [31], Javelin [32, 33], LFS [34], Legion [35], Globus [36], Entropia [37], and Jini [38] were developed as prototypes to explore general grid computing infrastructure. Charlotte and Javelin implemented a distributed memory model and a shared memory model, respectively, in a grid-computing environment. LFS, standing for Load Sharing Facility, is a product of Platform Computing aiming at efficient dynamic workload management. Entropia's DCGrid has the capability of scalable job management, providing system facilities for application job scheduling, deployment, and execution. The Globus project is probably the largest current academic project, with the goal of developing a basic software infrastructure for computations that "integrates geographically distributed computational and information resources." [39] The Globus grid programming toolkit in the Globus project designs and provides standard services for resource location and allocation, fault detection, executable management, and user authentication. Similar to Globus, Legion is an integrated grid-computing system with a set of standard grid-computing services. Jini uses Java middleware technology to support the general requirements of federating networking resources.

Among these grid-computing projects, the Globus project, which develops a software toolkit that includes software services and libraries for resource monitoring, discovery, management, and security control in creating and using a grid, has been adapted in more and more science and engineering projects using grid-computing facilities. Nowadays, the Globus toolkit becomes the "de facto" standard for grid computing.

As a result of the development of grid-computing projects, in 1998, several large grids across the United States have been deployed, and two test beds – Gusto [40] and Centurion [41] – have been used to test the Globus toolkit and Legion on real applications. Also, a year later, NASA deployed the Information Power Grid [42], and the US Department of Energy deployed ASCI DISCOM [43]. The European Grid Forum was established in the summer of 1999. More importantly, the fall of 2000 saw the US and European Grid Forum merge with the Asia-Pacific grid community, giving rise to the Global Grid Forum [44]. Through all these efforts, the physical infrastructure of grid computing has been well established.

Despite these encouraging developments, several fields require additional research and many problems still remain unsolved or require further improvements. These various grid projects may have different implementations with different emphasis for different problems. Nevertheless, all these projects concentrate on addressing the emerging issues in grid computing at the system level, the middleware level, or the programming level. The development of these projects is helpful for solving some of the problems in grid computing, such as, the heterogeneity of different computer systems and the interoperability of distributed resources, however, many issues, such as performance and trustworthiness requirements, still remain unsolved and require more research effort.

2.1.3 The General Working Paradigm in a Computational Grid

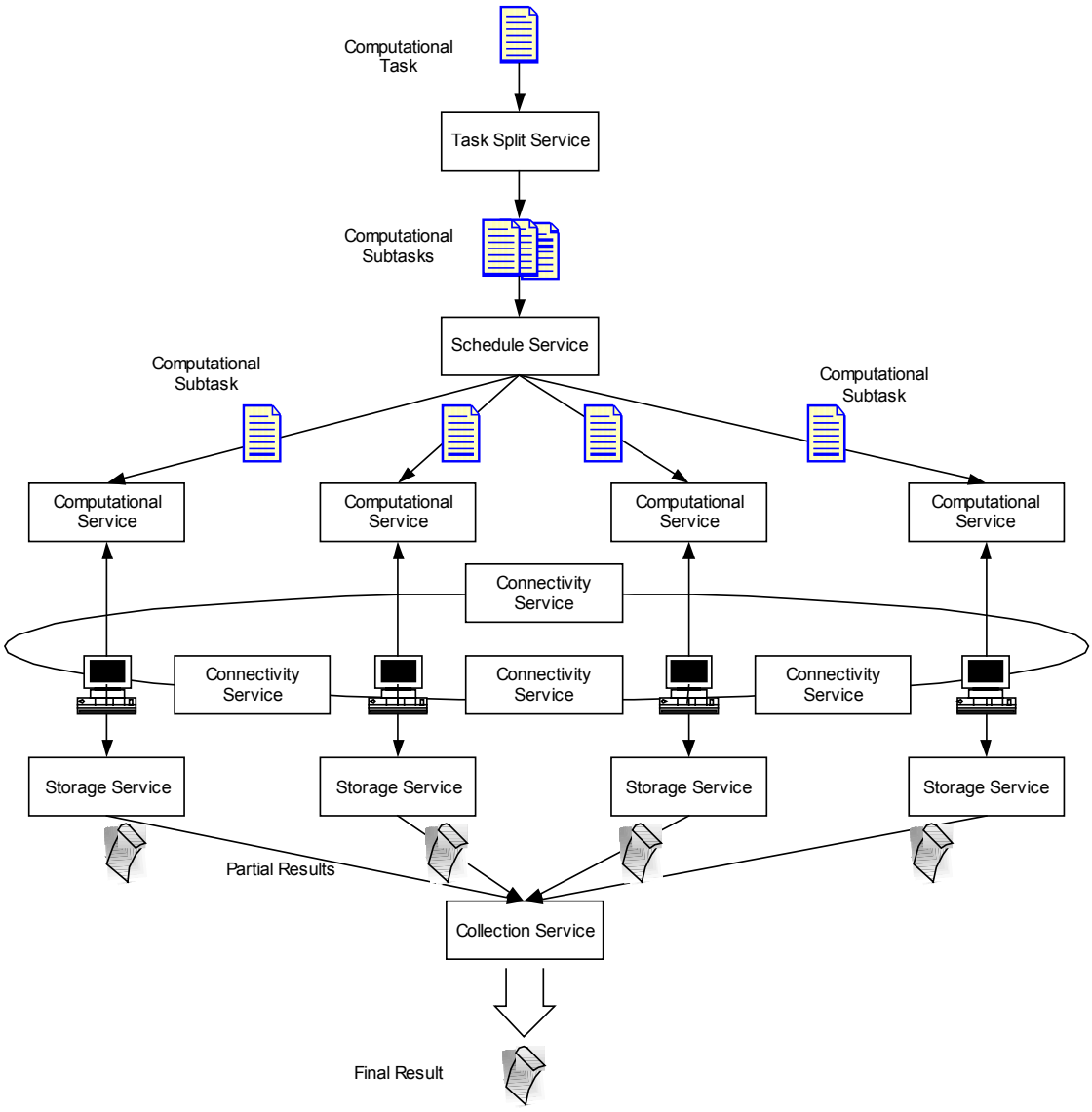

Figure 2.1 A General Working Paradigm of a Computational Grid

Although there are many existing grid computing systems, frameworks, and prototypes, they all have different implementations, while sharing a similar paradigm. In order to generalize a generic grid computing architecture, the Open Grid Services Architecture (OGSA) [45], which is being developed by IBM and the Globus project, treats all resources – CPU cycles, networks, storage, programs, databases – as services that provide some capability and try to define standard grid services. OGSA brings

together Web services standards, such as XML, WSDL, UDDI and SOAP, with the standards for grid computing developed by the Globus project. Through OGSA, the grid community has begun to specify open standards, Grid Computing protocols and Web services designed to enable large-scale cooperation and access to applications over public and private networks. Here is a list of services that are typically needed in a computational grid, although they may have different names in a particular system. Figure 2.1 shows a generic working paradigm for a computational grid operating with the cooperation of different services.

- **Task Split Service**: An agent running the task split service divides a big computational task into a number of relatively small subtasks.
- **Computational Service**: Computational service is provided by a computer or device in an organization that contributes CPU cycles for grid applications. Mechanisms that are required for starting programs and for monitoring and controlling the execution of the resulting processes are defined.
- **Connectivity Service:** The connectivity service provides connections between different resources. Core communication and authentication protocols required for grid-specific network transactions are defined.
- **Schedule Service**: The schedule service is responsible for distributing existing work units to available service providers. Resource tracking, task dispatching, and task monitoring mechanisms are defined.
- **Storage Service:** The storage service provides the capability of storing the program code repositories, data, intermediate results, and partial results during the processing of the grid computation.
- **Collection Service**: The collection service gathers the partial result data and assembles the final results.

In addition to the services mentioned above, there might be some other services needed in a computational grid, for example, an authentication service for a user to log on

and participate in a computation, a lookup service to locate a specific resource, a Web service to access the Web, and a statistical service to calculate resource usage.

2.1.4 Grid Computing Summary

The newly emerging grid computing technology has quickly attracted attention from science, engineering, and industry. In the above review, with an emphasis on the computational grid, we have reviewed the motivation of grid computing, current grid projects, and the grid architecture. Also, we have discussed potential issues in grid computing and various approaches to address these issues within existing grid projects.

Not every distributed computing application can fit in the grid-computing environment. The grid's dynamics and widely distributed characteristics welcome those applications that are robust and do not require much communication between distributed resources. The applications in most of the recent grid computing projects generally use an exhaustive search algorithm over a large data set. On the other end of the spectrum, there is a potentially large computational category of applications using the Monte Carlo method that are regarded as naturally parallel. These kinds of applications seem to be especially capable of taking advantage of the power of grid computing and therefore, are a potentially large computational category of grid applications. We will also discover that the nature of Monte Carlo applications can even be exploited to address the performance and trustworthiness issues emerging in grid computing at the application level. In the following section, we will review the developments and applications of the Monte Carlo method and explore the possibility of applying Monte Carlo applications within a grid-computing environment.

2.2 Monte Carlo Applications

Monte Carlo methods provide solutions to a variety of mathematical problems through statistical sampling. They are important techniques for performing simulation,

optimization and integration, and form the computational foundation for many fields including transport theory [6], quantum chromodynamics [7], and computational finance [46].

2.2.1 A Brief History of Monte Carlo Method

Perhaps the earliest documented use of the Monte Carlo method to find the solution to an integral is that of the Comte de Buffon. In 1777, he described the "Buffon Needle Experiment" [47] to estimate the value of π. Lord Kelvin appeared to have used random sampling to aid in evaluating some time integrals of the kinetic energy that appear in the kinetic theory of gases [48]. Later, many advances made in probability theory and the theory of random walks explored fields in which Monte Carlo methods can be used. In addition, Courant, Friedrichs, and Lewy showed the equivalence of the behavior of certain random walks to solutions of certain partial differential equations [49].

During the Second World War, the bringing together of such people as von Neumann, Fermi, Ulam, and Metropolis at the beginning of modern digital computers gave a strong impetus to the advancement of Monte Carlo [49]. E. Fermi in the 1930's used Monte Carlo in the calculation of neutron diffusion, and later designed the Fermiac, a Monte Carlo mechanical device used in the calculation of criticality in nuclear reactors. In the 1940's, a formal foundation for the Monte Carlo method was developed by von Neumann, who established the mathematical basis for probability density functions, inverse cumulative distribution functions, and pseudorandom number generators. The work was done in collaboration with S. Ulam [50], who realized the importance of the digital computer in the implementation of the approach. Papers appeared that described the new method and how it could be used to solve problems in statistical mechanics, radiation transport, economic modeling, and other fields [9].

Today's applications of Monte Carlo methods include: cancer therapy, traffic flow, "plug-drug" search, Dow-Jones forecasting, and oil well exploration, as well as more traditional physics applications like stellar evolution, reactor design, and quantum chromo-dynamics. Monte Carlo methods are widely used in the modeling of materials and chemicals [9], from grain growth modeling in metallic alloys, to the behavior of nanostructures, polymers, and protein structure prediction [8].

2.2.2 Distributed Monte Carlo Applications and the Grid

When using the Monte Carlo method, a set of computational random numbers has to be generated and used to statistically estimate a quantity of interest. The probabilistic convergence rate of this process is known to be approximately $O(N^{-1/2})$, where N is the number of underlying random samples [11]. A serious drawback of the Monte Carlo method is this slow rate of convergence. In order to accelerate the convergence rate of the Monte Carlo method, several techniques have been developed. Variance reduction methods, such as antithetic variates, control variates, stratification and importance sampling [11], reduce the variance, σ^2, which is a quantity that measures probabilistic uncertainty. Another method is applying the quasi-Monte Carlo method, which uses quasirandom (subrandom) sequences that are highly uniform (measured as "discrepancy" [51]) instead of the usual random or pseudorandom numbers. While pseudorandom numbers are constructed to imitate the behavior of truly random sequences, the members of a quasirandom sequence are deterministic, and achieve equidistribution often at the cost of correlation and independence. Therefore, quasi-Monte Carlo methods can often converge more rapidly, at a rate of $O(N^{-1}(\log N)^k)$, for some k [52]. Figure 2.2 shows the discrepancies of quasirandom numbers and pseudorandom numbers and the corresponding convergence rates of quasi-Monte Carlo and Monte Carlo for an application of evaluating an integral.

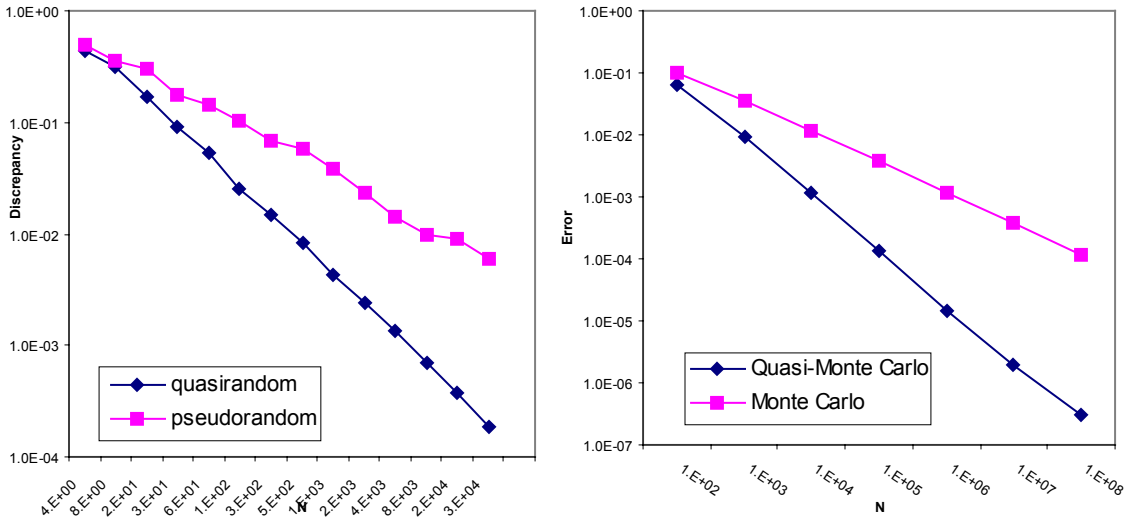

Figure 2.2 Discrepancy of Quasirandom Numbers and Pseudorandom Numbers and Convergence Rates of Quasi-Monte Carlo and Monte Carlo

Parallelism is an alternative 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. Due to this nature of Monte Carlo methods, Monte Carlo programs tend to compute as many samples as possible to reduce the statistical errors as much as possible. However, they consume vast computing resources and thus, are regarded as computation-bound. On the other hand, in a distributed Monte Carlo application, once a distributed task starts, it can usually be executed independently with the need for almost no inter-process communication. Therefore, Monte Carlo applications are widely perceived as computationally intensive but naturally parallel. The subsequent growth of computer power, especially that of parallel and distributed computers, has made large distributed Monte Carlo applications possible, with more ambitious calculations and accumulated knowledge learned from failures. Actually, many Monte Carlo applications and software packages in science and engineering, for example, CHARMM [53] for macromolecular dynamics simulation implemented using MPI, PMC [54] for nuclear physics simulation on the Livermore Message Passing System, MESYST [55] for simulation of 3D tracer dispersion in the atmosphere running on a CRAY T3E parallel machine, and MrBayes

[56] using Markov Chain Monte Carlo for Bayesian estimation of phylogeny on a Linux Beowulf cluster, have already taken advantage of the power of parallel systems to achieve a more accurate understanding of the problem or better performance of the computation.

Compared to the parallel computing environment, a large-scale distributed computing environment or a computational grid potentially has a tremendously large amount of computational power. Therefore, the grid has more unexplored potential and is expected to provide vast computational resources for distributed Monte Carlo computation. Several attempts have been made to use Monte Carlo applications in large-scale distributed computing paradigms. Y. Li and M. Mascagni developed "A Web-based Monte Carlo Integration Tool" (http://sprng.cs.fsu.edu/mcint/) [57] to set up a web-based application using the crude Monte Carlo method to implement multidimensional integration in the Condor distributed computing environment. M. Zhou built a Cycle Server to utilize the high throughput Condor system for a Brownian Langevin simulation of medical molecules [58]. F. Solms and W. H. Steeb used CORBA and Java to implement distributed Monte Carlo integration [59]. J. Basney et al.'s published paper "High Throughput Monte Carlo" [12] discussed using a High Throughput Computing system for distributed Monte Carlo applications. Entropia Inc. [37] is planning to apply Monte Carlo simulation applications to its DCGrid platform. Nowadays, with the development of grid-computing technology, we believe that Monte Carlo applications can be carried out on a much larger scale allowing one to extensively enhance the performance and accuracy of the Monte Carlo method.

Effectively exploring the power of distributed Monte Carlo applications requires that the underlying random number streams in each subtask be independent in a statistical sense. The main techniques used in parallel random number generators to distribute sequentially generated random number sequences among different processors include sequence splitting and leapfrog [60]. One problem with sequence splitting and the

leapfrog technique is that we must either assume that the number of parallel processes is fixed or at least bounded, which restricts the scalability of distributed Monte Carlo computations. Another technique to generate parallel random number sequences is to produce independent sequences by properly parameterizing pseudorandom number generators [61]. The SPRNG (Scalable Parallel Random Number Generators) library [62] 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 periods and good quality [63]. These generators meet the random number requirements of most Monte Carlo grid applications.

2.3 Conclusion

In this chapter, we provided a brief literature review of grid computing and Monte Carlo applications. For large-scale Monte Carlo analysis, "all the pieces of the puzzle" have just come into confluence. First of all, the distributed computing environment, especially the grid-computing environment with large-scale resource sharing and collaboration, is now sufficiently powerful to enable the simulation of very large engineering and physical systems. Secondly, with the emergence of scalable parallel random number generators, parallelization at any granularity appears to be easily implemented, and robust. Finally, many models, algorithms, and applications packages using Monte Carlo methods in various scientific and engineering areas have been well developed. All these motivate us to develop a high-performance and trustworthy grid-computing infrastructure to effectively and reliably deploy Monte Carlo computations on a computational grid.

CHAPTER 3

ANALYSIS OF GRID-BASED MONTE CARLO APPLICATIONS

Grid computing is characterized by large-scale sharing and cooperation of dynamically distributed resources. As mentioned in Chapter 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 [19], HARNESS [64], Javelin [32], Legion [35], Globus [36], and Entropia [37], concentrate on developing high-performance and trust-computing facilities through system-level approaches. In this chapter, we are going to develop application-level techniques to improve performance and enforce trustworthiness of grid-based Monte Carlo applications.

This chapter is organized as follows. In Section 3.1, we analyze the characteristics of Monte Carlo applications, which are a potentially large computational category of grid applications. Based on these analyses, in Section 3.2, we discuss improving the efficiency of the subtask-scheduling scheme by an "*N-out-of-M*" strategy and a Monte Carlo-specific lightweight checkpoint technique, which leads to a performance improvement for Monte Carlo grid computing. Also, in Section 3.3, we address the trustworthiness issues of Monte Carlo grid-computing applications by utilizing the statistical nature of Monte

Carlo partial results and by validating intermediate results utilizing the random number generator already in use in the Monte Carlo application. All these techniques eventually lead to a high-performance grid-computing infrastructure that is capable of providing trustworthy Monte Carlo computation services.

3.1 Introduction to Grid-based Monte Carlo Applications

Among 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, 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 the grid. 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 computationally intensive [65] and they tend to work on relatively small data sets while often consuming a large number of CPU cycles. As we mentioned before, parallelism is a way to accelerate the convergence of a Monte Carlo computation. Also, in a parallel/distributed Monte Carlo application, once a distributed task starts, it can usually be executed independently with almost no inter-process 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. Usually, these subtasks share the same program logic but are based on unique random number streams. Each subtask produces a partial result. Finally, the assembly of these partial results constitutes the final result of the whole Monte Carlo computation.

The intrinsically parallel aspect of Monte Carlo applications makes them an ideal fit for the grid-computing paradigm and motivates the use of the computational grid to effectively perform large-scale Monte Carlo computations. In general, grid-based Monte

Carlo applications can utilize the grid's *schedule service* to dispatch the independent subtasks to different nodes [66]. The execution of a subtask takes advantage of the *storage service* of the grid to store intermediate results and also 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. In Figure 3.1, we illustrate this generic paradigm for grid-based Monte Carlo applications on a computational grid. Furthermore, in the following sections of this chapter, we are going to show the technique of taking advantage of the inherent characteristics of Monte Carlo applications to reduce the wallclock time and to enforce the trustworthiness of the computation within the Monte Carlo grid-computing paradigm.

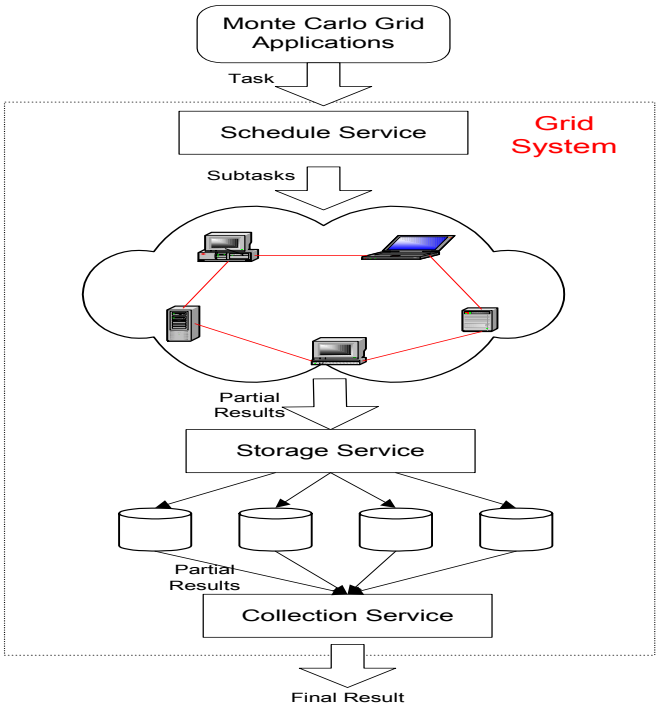

Figure 3.1: Monte Carlo Application on a Computational Grid

3.2 Improving the Performance of Grid-based Monte Carlo Computing

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

Before we discuss the *N-out-of-M* strategy, we first analyze the generic computational replication technique to enhance performance on a computational grid. Then, using the outcome analytical model of computational replication on the grid as a theoretical foundation, we extend the generic computational replication technique to the *N-out-of-M* subtask schedule strategy specifically for high-performance grid-based Monte Carlo applications.

3.2.1.1 Computational Replication on the Grid

A large-scale computational grid can, in principle, offer a tremendous amount of low-cost computational power. This attracts many computationally intensive scientific applications. On the other hand, significant challenges also arise. Within a computational grid's dynamic environment, the computational capabilities of each node vary greatly. A node might be a high-end supercomputer, or a low-end personal computer, even just an intelligent widget. As a result, a task running on different nodes on the grid will have a huge range of completion times. Also, due to unreliable network connections and the possible unavailability of a node, an executing task may be delayed or even halted at any time. Therefore, from the grid-application point of view, how quickly a computational grid can complete a group of submitted tasks from an application becomes an issue of prime importance.

In this section, we investigate a computational replication technique [67] to develop an optimal scheduling mechanism to improve the throughput of a computational grid and reduce task completion time. This is different from the task-scheduling problem that has been discussed for many conventional parallel or distributed computing environments [68, 69], where there are a very limited number of nodes or processors. In contrast, on a computational grid, the available computational service providers can

essentially be treated as unlimited compared to the number of existing tasks. Therefore, we have more freedom to use these massive computational resources as trade-offs to achieve better task completion times.

3.2.1.1.1 Introduction to Computational Replication

Replication is a well-known technique for improving availability in an unreliable system. In fault-tolerant computing, replication is also a technique to overcome faults [70, 71]. The replication technique has already been favorably utilized in grid computing. In SETI@home [3], a majority voting mechanism[1] is applied to check the correctness of a task. Ranganathan, Iamnitchi, and Foster discussed using dynamic model-driven replication to obtain high data availability in a large peer-to-peer community [72]. In this paper, we are interested in improving the performance of a computational grid by replicate scheduling of grid tasks.

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

The implementation of computational replication on a computational grid is rather simple. Figure 3.2 shows the mechanism of replicate scheduling in a grid-computing

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

environment. When the computational grid receives a task, *r* copies are replicated and scheduled to *r* different nodes. At that point in time, *r* copies of the task are concurrently running. Once an execution is complete and the corresponding result is obtained, the task is regarded as finished. Termination signals can be sent to the other nodes to abort their current running jobs.

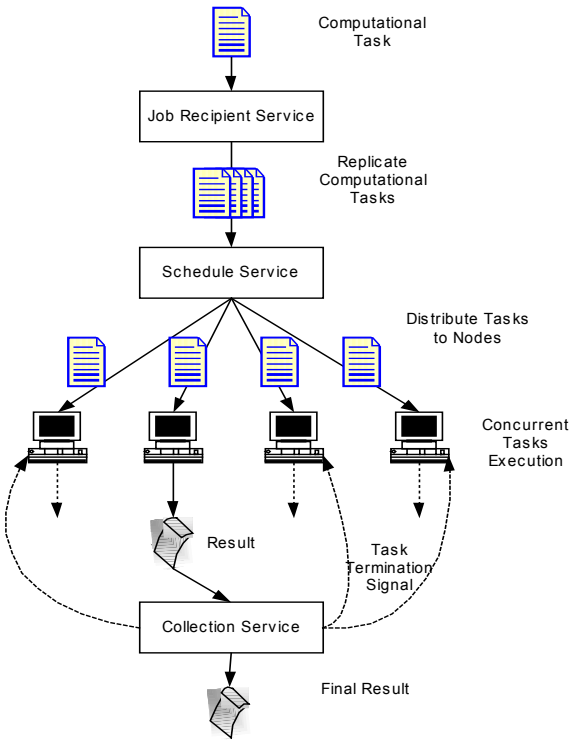

Figure 3.2: Replicate Scheduling in a Computational Grid

Using the computational replication technique can prevent slow or unstable nodes from slowing down or halting a grid task with high probability, which could lead to a reduced completion time of this task. However, we do not wish to imply that the more replicas, the better. The execution of too many copies of a task may not contribute much to reducing completion time but may significantly increase the grid system's workload. Such problems can be found in the metacomputing prototype Charlotte [31] using its eager scheduling mechanism. Eager scheduling aggressively assigns and reassigns

existing tasks to available nodes in the distributed-computing system to keep all the nodes busy. Nevertheless, the following phenomenon may occur: there may be many copies of a task running on the system and occupying many computational resources. However, the later arriving tasks may not be able to find an available node, which will lead to reducing the system throughput. In short, to determine what is a "reasonable" number of replicas becomes critical for the computational replication technique. We will establish a system model to probe for answers to this question in the next section.

3.2.1.1.2 Analytical Model of Computational Replication

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

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

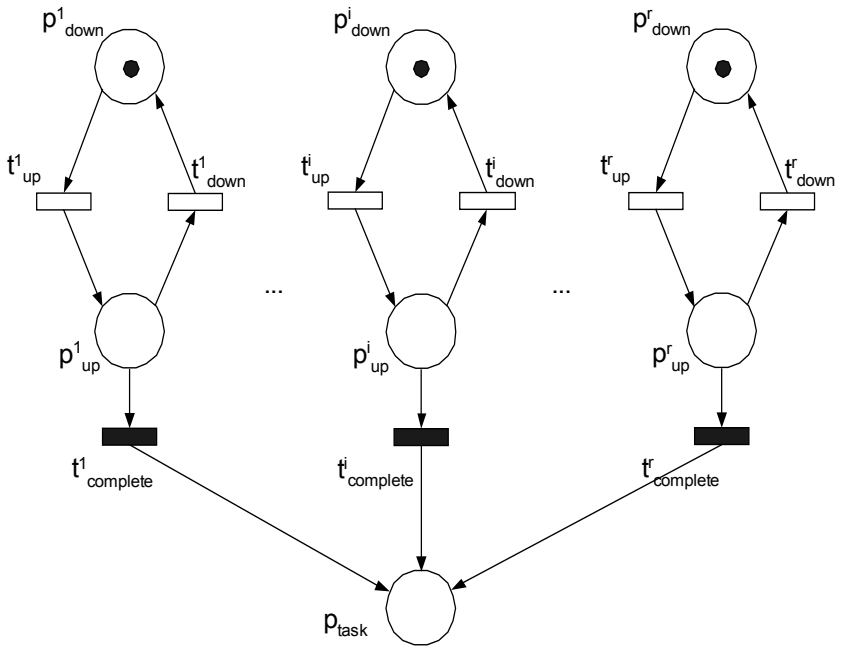

Figure 3.3: Petri Net Modeling of Computation Replication in the Grid

Figure 3.3 shows the Petri Net (PN) model of replicated tasks concurrently running on a computational grid. In this PN model, a node, *i*, alternates between an up state (place p^i_{up}) and a down state (place p^i_{down}). Transition t^i_{down} represents node unavailability (with unavailability rate λ) and transition t^i_{up} node back to service (with availability rate μ). Transition $t^i_{complete}$ is assigned the task progress threshold W (usually 100%) so that the task completion condition (token in p_{task}) is reached when W is hit.

Let r be the total number of computing replicas,

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

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

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

$$S_i(t) = \Pr(T_i \geq t) = \int_t^{\infty} \theta_i e^{-\theta_i x} dx = e^{-\theta_i t} \quad (1)$$

.

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

$$p_{task_i}(t) = 1 - S_i(t) \quad (2)$$

.

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

$$p_{replica}(t) = 1 - \Pr(Min(T_1, T_2, ..., T_r) \geq t) = 1 - e^{-\sum_{i=1}^{r} \theta_i t} \quad (3)$$

.

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

$$r = \left\lceil -\frac{\ln(1-\alpha)}{\theta t} \right\rceil. \quad (4)$$

3.2.1.1.3 Simulation Results of Generic Computational Replication on the Grid

In our simulation program, we simulated a 1,000-node computational grid. Nodes join and leave the grid 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 about $1/\theta$ time steps to compute, e.g., a fast node with service rate 0.01 will take 100 time steps to complete the task on average 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 computational replication technique with the dynamic rescheduling technique.

3.2.1.1.3.1 Model Validation

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

Figure 3.4 shows the comparison between the simulation results and the prediction from our analytical model. From the graph, we can see that the actual behavior matches our model prediction quite well. Also, we notice that with 1 task running, at least 600 time steps are required to obtain a 90% task completion percentage, however, with 4 replicas, less than 200 time steps are required to obtain this same percentage. This indicates a significant task completion time reduction using the computational replication technique. However, we also found that even when we increase the number of current tasks to 20, we cannot significantly further increase task performance. Therefore, with a proper number of replicas, we can achieve an optimal performance/cost ratio.

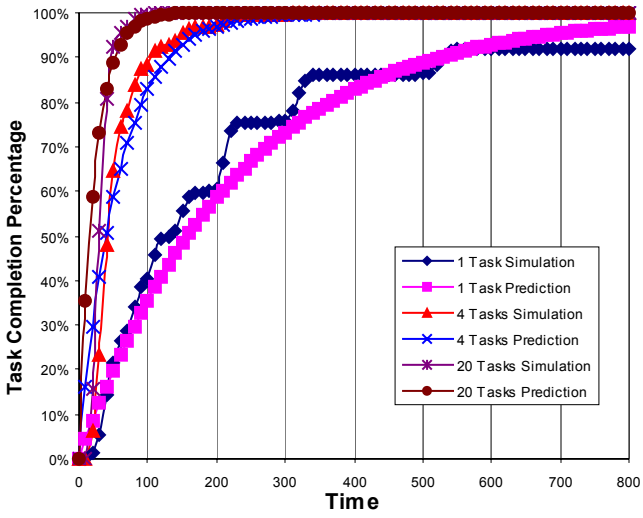

Figure 3.4: Model Prediction versus Actual Behavior with Different Numbers of Replicas

3.2.1.1.3.2 Replicate Scheduling vs. Dynamic Rescheduling

To prevent a slow node from delaying or halting the completion of a grid task, the dynamic rescheduling technique is another popular method used in existing computational grid systems like Condor [19, 20] and Entropia [37]. In dynamic rescheduling, the system keeps track of the execution of each task. When a task is halted, a checkpoint of the execution is then generated. Next, the schedule service will look for another available and appropriate node to reschedule the task. After the checkpoint data file is transferred to the new node, the task then continues to execute on the new node by recovering the execution process of the task. The dynamic rescheduling technique can keep the task running all the time but at the cost of additional system administration and rescheduling overhead involving task status monitoring, checkpointing, searching for available nodes, network transferring, task rescheduling, and execution recovering.

We simulate the scheduling mechanism using the dynamic rescheduling technique. When a node running a task is down, the task is rescheduled to another available node. The rescheduling penalty is taken into consideration when task rescheduling occurs in the simulation. Figures 3.5 and 3.6 illustrate the task completion

time comparison between replicate scheduling and dynamic rescheduling. The data in Figure 3.5 come from simulation on a computational grid comprised of nodes with similar performance characteristics. This can be a grid constructed from computers in a computer lab that have similar performance parameters and are connected by a high-speed network. The cost of task rescheduling is relatively low in such a situation. Our simulation results show that the dynamic rescheduling technique has a better task completion time than that of replicate scheduling when the node unavailability rate is high. When the node down rate is low, both techniques have a similar simulated performance. Figure 3.6 simulates a computational grid whose nodes have computational capabilities in a wide range. In practice, this grid can be a system with geographically widely distributed nodes like SETI@home [3]. In this grid system, a node might be a high-end supercomputer, or a low-end personal computer, or even just an intelligent widget. The connection among nodes is via a low-speed network, which carries a high task rescheduling cost. We notice that in our simulation results, replicate scheduling using an appropriate number of replicas has a better task completion time than that of the dynamic rescheduling technique.

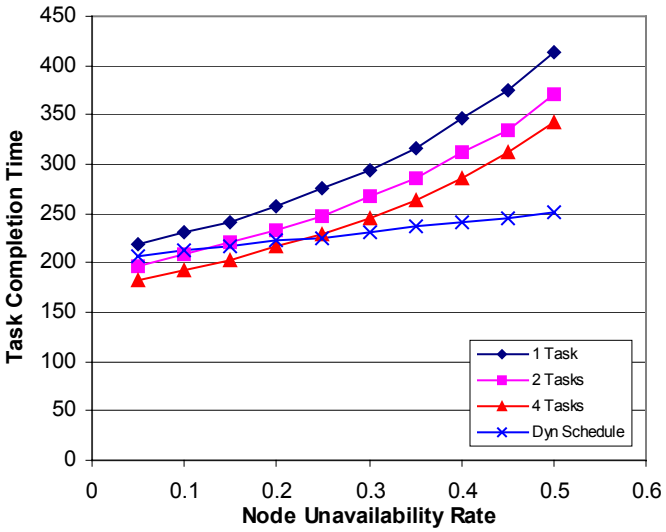

Figure 3.5: Replicate Scheduling vs. Dynamic Rescheduling on a Computational Grid with Nodes Sharing Similar Performance Parameters

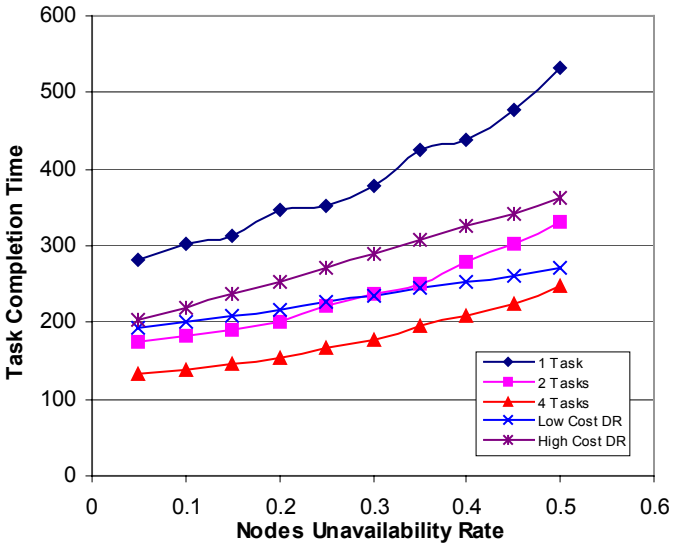

Figure 3.6: Replicate Scheduling vs. Dynamic Rescheduling on a Computational Grid with Nodes having a Wide Range of Computational Capabilities

3.2.1.2 The *N-out-of-M* Strategy for Monte Carlo Applications

In this section, we are interested in extending the computational replication technique to grid Monte Carlo applications to improve their performance. In the typical execution of a Monte Carlo computation on a grid system, we split the entire computational task into *N* subtasks, with each subtask based on unique independent random number streams. We then schedule each subtask onto the nodes in the grid system. In this case, the assembly of the final result requires all of the *N* partial results generated from the *N* subtasks. In this situation, each subtask is a "key" subtask, since the suspension or delay of any one of these subtasks will have a direct effect on the completion time of the whole task. To address this problem, system-level methods are used in many grid or distributed-computing systems. For example, Entropia [37] 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. To address this issue at the application level, we can use the so-called *N-out-of-M* subtask scheduling strategy specific for grid-based Monte Carlo applications, which is an extension of the computational replication technique.

3.2.1.2.1 Introduction to N-out-of-M Subtask Scheduling

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

When we are running Monte Carlo applications, what we really care about is how many random samples (random trajectories) we must 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.

Figure 3.7 shows an example of a distributed Monte Carlo computation using the "*6-out-of-10*" strategy. In this example, 6 partial results are needed and 10 subtasks are actually scheduled. During the computation, one subtask is suspended for some unknown reason. In addition, some subtasks have very short completion time while others execute

very slowly. However, when 6 of the subtasks are complete, the whole computation is complete. The suspended subtask and the slow subtasks do not affect the completion of the whole computational task.

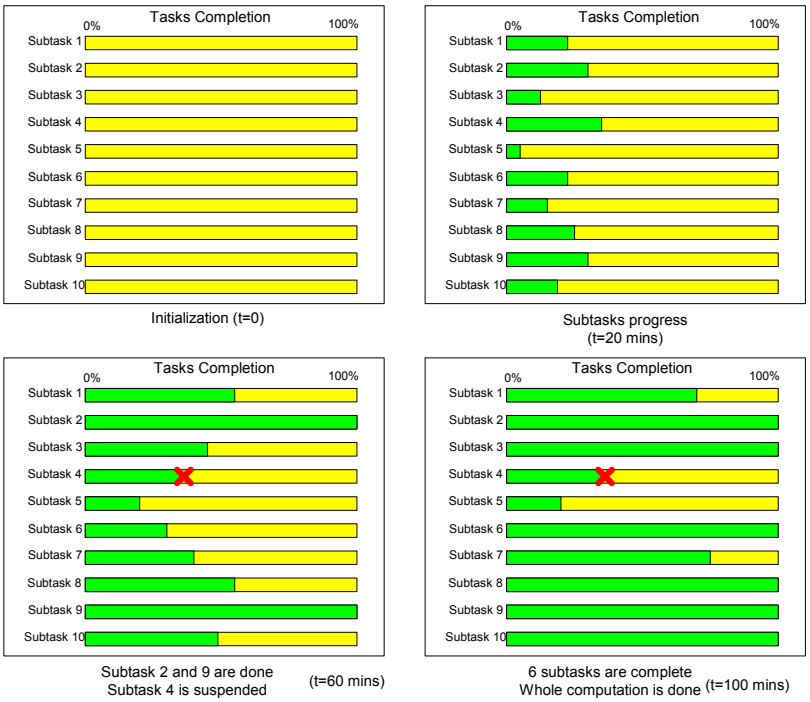

Figure 3.7 Example of the "*6-out-of-10*" Strategy with 1 Suspended and 3 "Slow" Subtasks

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.

3.2.1.2.2 The Binomial Model of the N-out-of-M Strategy

In Monte Carlo applications, N is determined by the application and it 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. At the same time, we make similar assumptions as those in Subsection 3.2.1.1.2 to set up our model.

Figure 3.8 shows the PN model of the *N-out-of-M* subtask schedule strategy. Similar to the PN model of generic computational replication, in each individual node, i, places p^i_{up} and p^i_{down}, transitions t^i_{down}, t^i_{up}, and $t^i_{complete}$ have the same meaning and parameters as the ones in Figure 3.3. This PN model has M such nodes in total. When $p_{subtask}$ gathers N tokens, transition $t_{N\text{-}out\text{-}of\text{-}M}$ enables us to fire and a token in place $p_{complete}$ indicates the completion of the Monte Carlo task.

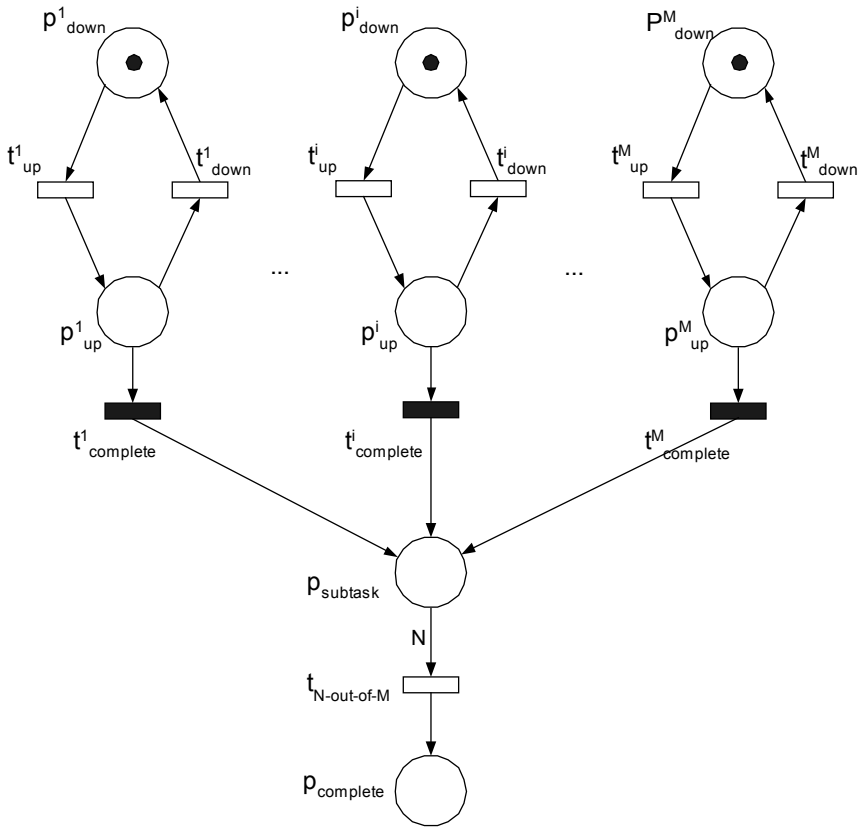

Figure 3.8 Petri Net Modeling of *N-out-of-M* Subtask Schedule Strategy

We establish a 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)*. *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 $p^i_{sys} = \mu/(\mu+\lambda)$, similarly to the computational replication model, and

θ_i' be the service rate of node i, which can be measured as the number of tasks that can be finished within a specific period of time without interruption.

Considering 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

$$p(t) = \frac{1}{S}\sum_{i=1}^{S}(1-e^{\theta_i t}) = 1-\frac{1}{S}\sum_{i=1}^{S}e^{\theta_i t} . \quad (5)$$

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

$$p(t) = 1 - \frac{1}{L}\int_0^L e^{\phi(\theta)t} d\theta . \quad (6)$$

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

$$p(t) = 1 - e^{\theta t} . \quad (7)$$

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

$$P_{Exactly-N-out-of-M}(t) = \binom{M}{N} p^N(t) \times (1 - p(t))^{M-N} . \quad (8)$$

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

$$P_{Exactly-N-out-of-M}(t) \approx \frac{\lambda^M}{M!} e^{-\lambda} . \quad (9)$$

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

$$P_{N-out-of-M}(t) = \sum_{i=N}^{M} \binom{M}{i} p^i(t) \times (1 - p(t))^{M-i} . \quad (10)$$

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

$$P_{N-out-of-N}(t) = p^N(t). \quad (11)$$

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 $P_{N\text{-}out\text{-}of\text{-}M}(t)$ until the value of $P_{N\text{-}out\text{-}of\text{-}M}(t)$ is greater than α. This empirically gives us the minimum value of M. 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

$$\Phi(\frac{M-m}{\sigma}) - \Phi(\frac{N-m}{\sigma}) \geq \alpha, \quad (12)$$

where Φ is the unit normal cumulative density function.

In a grid system, nodes providing computational services join and leave dynamically. Some nodes are considered "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.

3.2.1.2.3 The Simulation of the N-out-of-M Strategy

In our simulation program of the *N-out-of-M* strategy, we simulated a computational grid with the same behavior as the one described in Subsection 3.2.1.1.3 when we perform simulation of the computational replication technique. Our goals are to

1) evaluate the validity of our binomial model, and to

2) compare the performance of the *N-out-of-M* strategy in grid systems with different configurations.

Figure 3.9 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 a proper choice of *M* (20 in the graph), the Monte Carlo task completion time can be improved significantly over the *N-out-of-N* strategy. However, if we enlarge *M* too much, the workload of the system increases without significantly reducing the Monte Carlo task completion time. 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.

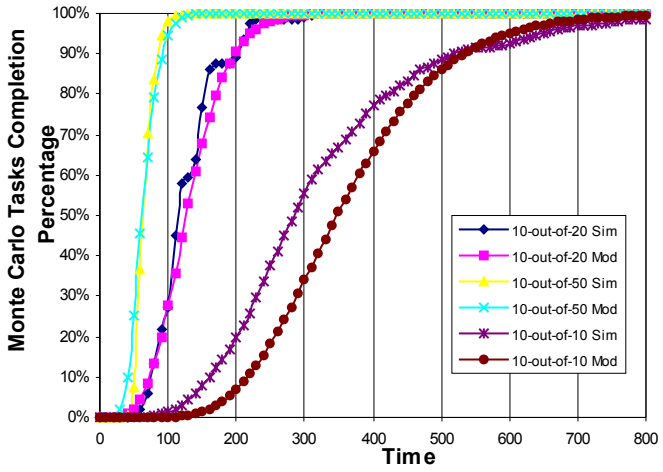

Figure 3.9 Simulations and Model Prediction of the *N-out-of-M* Scheduling Strategy for Grid Monte Carlo Applications

Figure 3.10 and 3.11 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 3.10 and 0.003 in Figure 3.11). Figure 3.10 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 3.11 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 [2], 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, which is exactly the subtask completion time for a single node with 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!

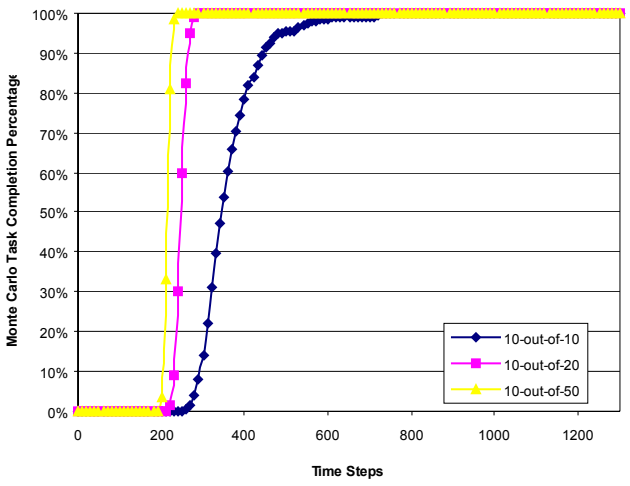

Figure 3.10 Simulations of the *N-out-of-M* Strategy on a Grid System with Nodes Service Rates Normally Distributed (Mean=0.005, Variance=0.001)

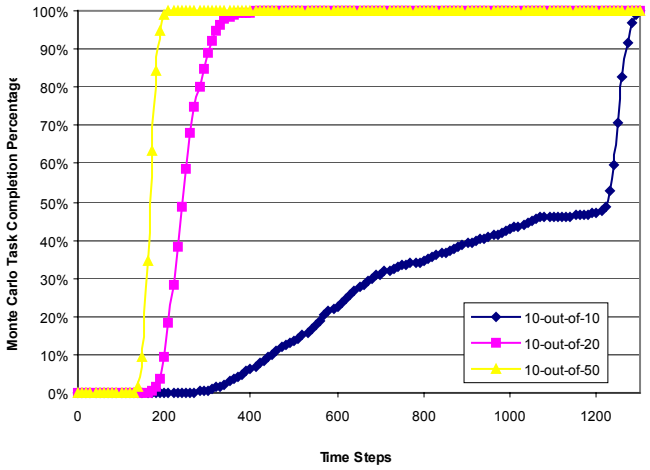

Figure 3.11 Simulations of the *N-out-of-M* Strategy on a Grid System with Nodes Service Rates Normally Distributed (Mean=0.005, Variance=0.003)

3.2.2 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 [20]. 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.

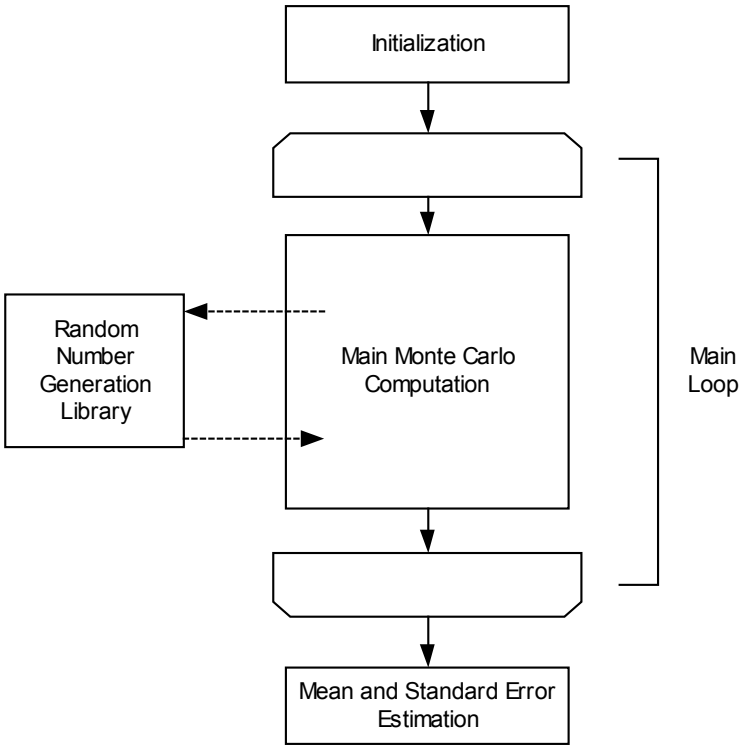

Figure 3.12 A Typical Programming Structure for a Monte Carlo Application

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 different Monte Carlo applications may have very different implementations, many of them can be developed or adjusted in a typical programming structure shown in Figure 3.12.

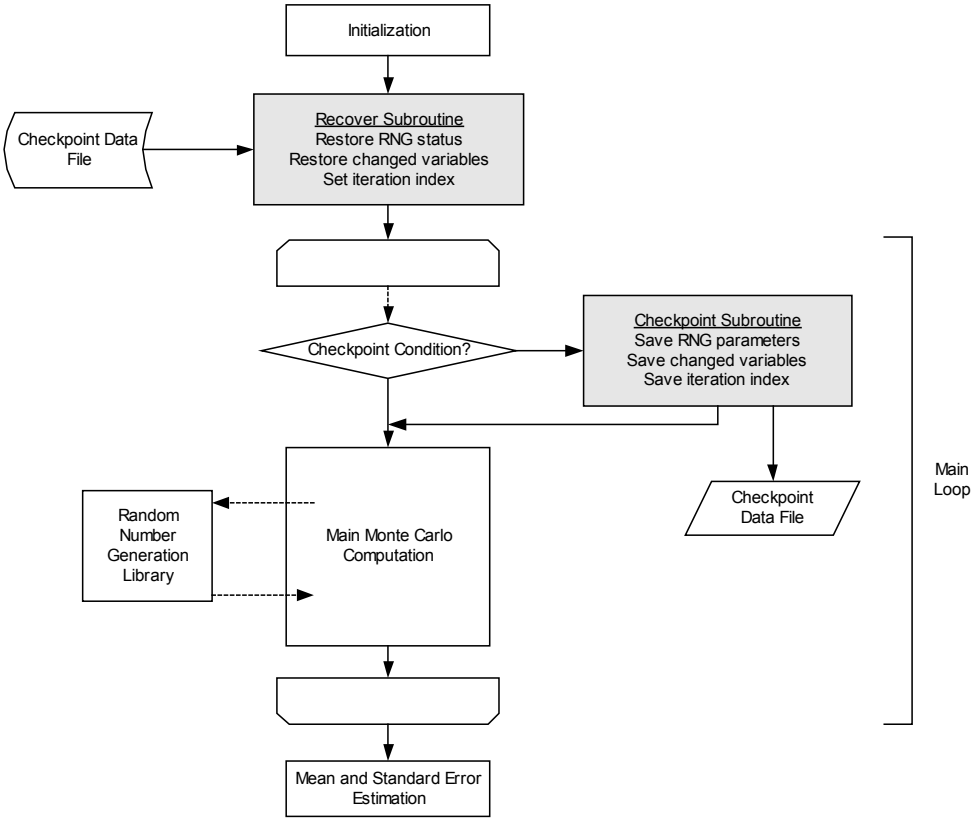

Figure 3.13 A Monte Carlo Application with Checkpoint and Recovery Facilities

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 [73] 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 to the one shown in Figure 3.13. However, the implementation of application level checkpointing will somewhat increase the complexity of developing new Monte Carlo grid applications.

3.3 Enhancing the Trustworthiness of Grid-based Monte Carlo Computing

3.3.1 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 [3] has shown that users sometimes 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.

The following example, Example 3.1, illustrates how an erroneous computational partial result effects the whole computation. Let us consider the following hypothetical Monte Carlo computation. Suppose we wish to evaluate integral

$$\int_0^1 \cdots \int_0^1 \frac{4x_1 x_3^2 e^{2x_1 x_3}}{(1+x_2+x_4)^2} e^{x_5+\ldots+x_{20}} x_{21} x_{22} \ldots x_{25} dx_1 \ldots dx_{25} \ . \qquad (13)$$

The exact solution to 8-digits of this integral is 103.81372. In the experiment, we plan to use crude Monte Carlo on a grid system with 1,000 nodes.

Table 3.1 Hypothetical Partial Results of Example 3.1

Subtask #	Partial Results
1	103.8999347
2	104.0002782
3	103.7795764
4	103.6894540
…	
561	89782.048998
…	
997	103.9235347
998	103.8727823
999	103.8557640
1000	103.7891408

Table 3.1 tabulates the partial results from volunteer computers. Due to an error, the partial result returned from the node running subtask #561 is clearly bad. The fault may have been due to an error in the computation, a network communication error, or malicious activity, but that is not important. The effect is that the whole computational result ends as 193.280805, considerably off the exact answer. From this example, we see that, in Monte Carlo grid computing, the final computational result may be sensitive to each of the partial results obtained from nodes in the grid system. An error in a computation may seriously hurt the whole computation.

To enforce the correctness of the computation, many distributed computing or grid systems adapt fault-tolerant methods, like duplicate checking [74] and majority vote [75]. 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, 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

$$\hat{f} = \frac{1}{n}\sum_{i=1}^{n} f_i , \qquad (14)$$

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

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} (f_i - \hat{f})^2} . \qquad (15)$$

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. 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 based on all the partial results. Suppose we are running *n* Monte Carlo subtasks on the grid, the *i*th subtask will eventually return a 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 the 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.

An alternative way of Monte Carlo partial result validation requires the existence of a trusted grid node. In this method, the computation of a subtask is executed on the trusted node and its partial result is used to validate the partial results from the other subtasks from untrusted nodes. Since the Monte Carlo subtask on the trusted node also evaluates a certain amount of random samples, the partial result of this subtask also has a mean f_t and standard deviation σ_t. According to Tchebycheff's inequality,

$$\Pr ob\{(f - f_t) \geq k\sigma_t\} \leq \frac{1}{k^2}, \qquad (16)$$

the other partial results should be in the interval of $[f_t - k\sigma_t, f_t + k\sigma_t]$ with probability *1 - 1/k²*. Therefore, suppose *n* subtasks are calculated, we can select a proper value of *k*, where

$$k = \lfloor \sqrt{cn} \rfloor, \qquad (17)$$

to build a confidence interval with a confidence of $\alpha\% = 1 - 1/cn$. Again, here c is a subjective constant that we flag events that should only occur once in a group of size cn. Then, the confidence interval can be used to validate each partial result, f_i, from the untrusted grid nodes. Practically, this interval may be much bigger than the one based on all the partial results. However, one benefit of the method based on the trusted node is, once we obtain the confidence interval, we can use it to validate each partial result returned from a grid node immediately and do not need to wait until all the required partial results are ready. This especially fits the *N-out-of-M* scheduling strategy discussed in Section 3.2.1.

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.

Let us now come back to the previous example, Example 3.1. If a hypothetical partial result happens as the one (#561) in Example 3.1, the outlier lies 30 standard deviations to the right of the mean. As we know from calculating the confidence interval, we have $\alpha = 99.9999999999\%$ within 7 standard deviations. A outlier falling outside of 7 standard deviations of the mean will be expected to happen by chance only once in 10^9 experiments. Therefore, the erroneous partial result of #561 in Example 3.1 will easily be captured and flagged as abnormal.

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 computations is very small.

3.3.2 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 [3], 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 an inexpensive 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.

Some pseudorandom number generators exhibit the fast leap-ahead property [76], which enables us to easily and economically jump ahead in the sequence. By the fast leap-ahead algorithm, we can transform a seed at a particular point in a pseudorandom number generator's cycle to a new point n steps away in $O(\log_2 n)$ "operations," where one "operation" is the cost of generating a single random number. Figure 3.14 shows the leaping in a random number sequence. For example, to linear congruential generators (LCGs) in the general form,

$$X_i = (a * X_{i-1} + c) (\text{mod} \quad M), \tag{18}$$

where a is known as the multiplier, M is the modulus, and c is the additive constant, the corresponding leapfrog generator with leaping length n can be represented as

$$X_i = (A * X_{i-1} + C) (\text{mod} \quad M) \tag{19}$$

by replacing the multiplier a and the additive constant c by new values A and C, where

$$\begin{aligned} A &= a^n (\text{mod} \quad M), \textit{ and} \\ C &= ca^{n-1}(a-1)^{-1} (\text{mod} \quad M) \end{aligned}. \tag{20}$$

It requires $O(\log_2 n)$ "operations" to produce the parameters of a leapfrog generator. After this, the new leapfrog generator can produce random number sequence leaping by distance n in the sequence generated by the original generator. The generation of each leaping random number costs only a unit "operation." The fast leap-ahead algorithm can also be implemented in combined LCGs [77], lagged Fibonacci generators (LFG) [76], and shift-register generators (SRG) [78]. The description of these algorithms can be found in Appendix A of this dissertation.

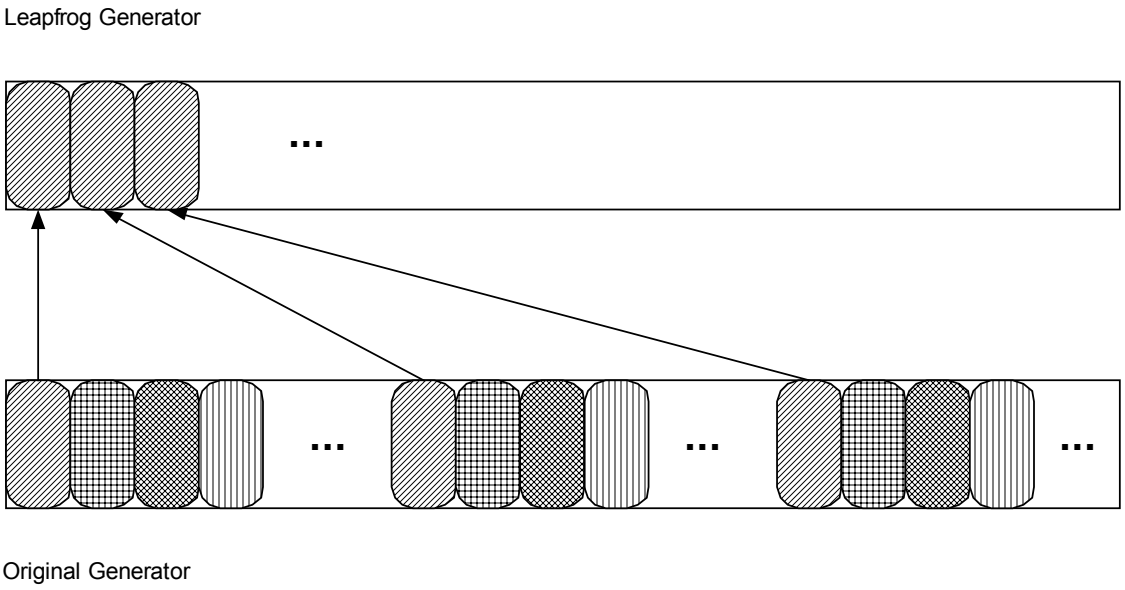

Figure 3.14 Fast Leap-ahead Property of a Pseudorandom Number Generator

Usually, the fast leap-ahead property of the pseudorandom number generators is widely used to spread a serial pseudorandom number sequence across parallel processors. Here, we use the fast leap-ahead technique to economically regenerate any selected pseudorandom numbers used in a Monte Carlo subtask. These selected pseudorandom numbers are used as the intermediate values for further validation. 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 re-compute the Nth, $2N$th, …, kNth random numbers applying the corresponding leapfrog generator with the same seed as used in this subtask. We then simply match them. A mismatch indicates problems during the execution of the subtask.

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 the intermediate value checking technique is certainly not a perfect way to ensure correctness and trustworthiness, a user determined to fake 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.

3.4 Summary of Analysis of Grid-based Monte Carlo Applications

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 computational grid to implement large-scale grid-based Monte Carlo computing. More importantly, based on our analysis of the statistical nature of Monte Carlo applications and the cryptographic nature of random numbers, we developed techniques to enhance the performance and trustworthiness of grid-based Monte Carlo computations at the application level. In the next chapter, we are going to elucidate the implementation details of a grid-computing infrastructure for Monte Carlo applications utilizing these techniques.

CHAPTER 4

GCIMCA: A GLOBUS AND SPRNG IMPLEMENTATION OF A GRID-COMPUTING INFRASTRUCTURE FOR MONTE CARLO APPLICATIONS

In Chapter 3, we analyzed the inherent characteristics of Monte Carlo applications and the underlying random number generators to develop techniques for large-scale grid-based Monte Carlo computations. Based on these techniques, in this chapter, we are going to discuss the implementation of a software toolkit, which we refer to as the Grid-Computing Infrastructure for Monte Carlo Application (GCIMCA) [79], to provide high-performance and trustworthy grid services for grid-based Monte Carlo computations.

In addition to the approaches and techniques we discussed in Chapter 3, the implementation of GCIMCA 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 [39] facilitates the creation and utilization of a computational grid for large distributed computational jobs. The Scalable Parallel Random Number Generators (SPRNG) [63] library is designed to generate practically an 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, 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.

Chapter 4 is organized as follows. We provide a brief introduction to the Globus software toolkit and the SPRNG library in Section 4.2 and 4.3, respectively. In Section 4.4, we elucidate the architecture of GCIMCA. Section 4.5 provides an illustration of the working paradigm of GCIMCA. The detailed implementations of the core services and facilities in GCIMCA are discussed in Section 4.6.

4.1 Introduction to the Globus Toolkit

The Globus project is a joint effort of the Argonne National Laboratory, the University of Chicago, and the University of Southern California. The Globus toolkit is the concrete result of the Globus project, which is a symbiotic set of basic grid services, including resource management, security protocols, information services, communication services, fault tolerance services, and remote data access facilities. It allows developers to easily build computational grid infrastructure for grid-based applications and use geographically distributed resources on the grid [80].

The fabric of the grid comprises the underlying computers, operating systems, networks, storage systems, and routers. The Globus toolkit provides grid services and facilities integrating the components of the grid fabric. These grid services and facilities include:

- GRAM (the Globus Resource Allocation Manager) – a basic library service that provides capabilities to do remote-submission job start-up,
- GIS (the Grid Information Service) – information service to discover the properties of the computers, systems, and networks that a user expects to use,
- GSI (the Grid Security Infrastructure) – a library for generic security services for applications running on the grid, and

- GASS (the Global Access to Secondary Storage) – a service for primitive access to remote data distributed on the grid.

With these grid services and facilities provided by the Globus toolkit, a grid application is capable of having uniform access to distributed resources with diverse scheduling mechanisms; using information services for resource publication, discovery, and selection; utilizing grid-based APIs for remote file management, staging of executables and data; and achieving enhanced performance through multiple communication protocols.

4.2 Introduction to the SPRNG Library

Development of the SPRNG library is a research project funded by the U.S. Department of Energy at the Florida State University Department of Computer Science. The goal of the SPRNG project is "to develop, implement, and test a scalable package for parallel pseudorandom number generation that will be easy to use on a variety of architectures, especially for large-scale parallel Monte Carlo applications." [63]

SPRNG is designed to use parameterized pseudorandom number generators to provide independent random number streams for parallel processes. The following generators are the core generators available in SPRNG:

- Additive lagged-Fibonacci generators (LFG),
- Multiplicative lagged-Fibonacci generators (MLFG),
- Prime modulus multiplicative congruential generators (PMLCG),
- Power-of-two modulus linear congruential generators (LCG), and
- Combined multiple recursive generators (CMRG).

In addition, Shift-Register Generators (SRG) and Inversive Congruential Generators (ICG) are currently being implemented to integrate with the existing SPRNG generators. Grid-based Monte Carlo applications require large number of parallel random number streams with good quality. Some generators in the SPRNG library can provide up to

2^{78000} – 1 independent random number streams 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 grid-based Monte Carlo applications.

4.3 Architecture of GCIMCA

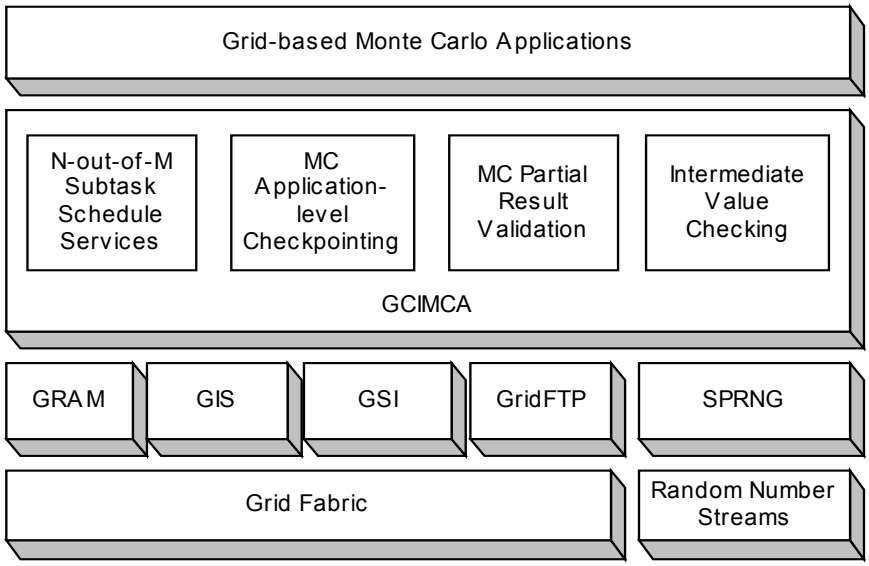

Figure 4.1 Architecture of GCIMCA

GCIMCA is designed on the top of the grid services provided by Globus, [39] and supplies facilities and services for grid-based Monte Carlo applications. Figure 4.1 shows the architecture of GCIMCA. 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 grid nodes with proper configurations for running a Monte Carlo subtask. GSI offers security services such as authentication, encryption and decryption for running Monte Carlo applications on the grid. GridFTP provides a uniform interface for application 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. With the services in GCIMCA, compared to the generic diagram of a grid application in Figure 2.1 we described in Chapter 2, the diagram of grid-based Monte Carlo computation is shown in Figure 4.2.

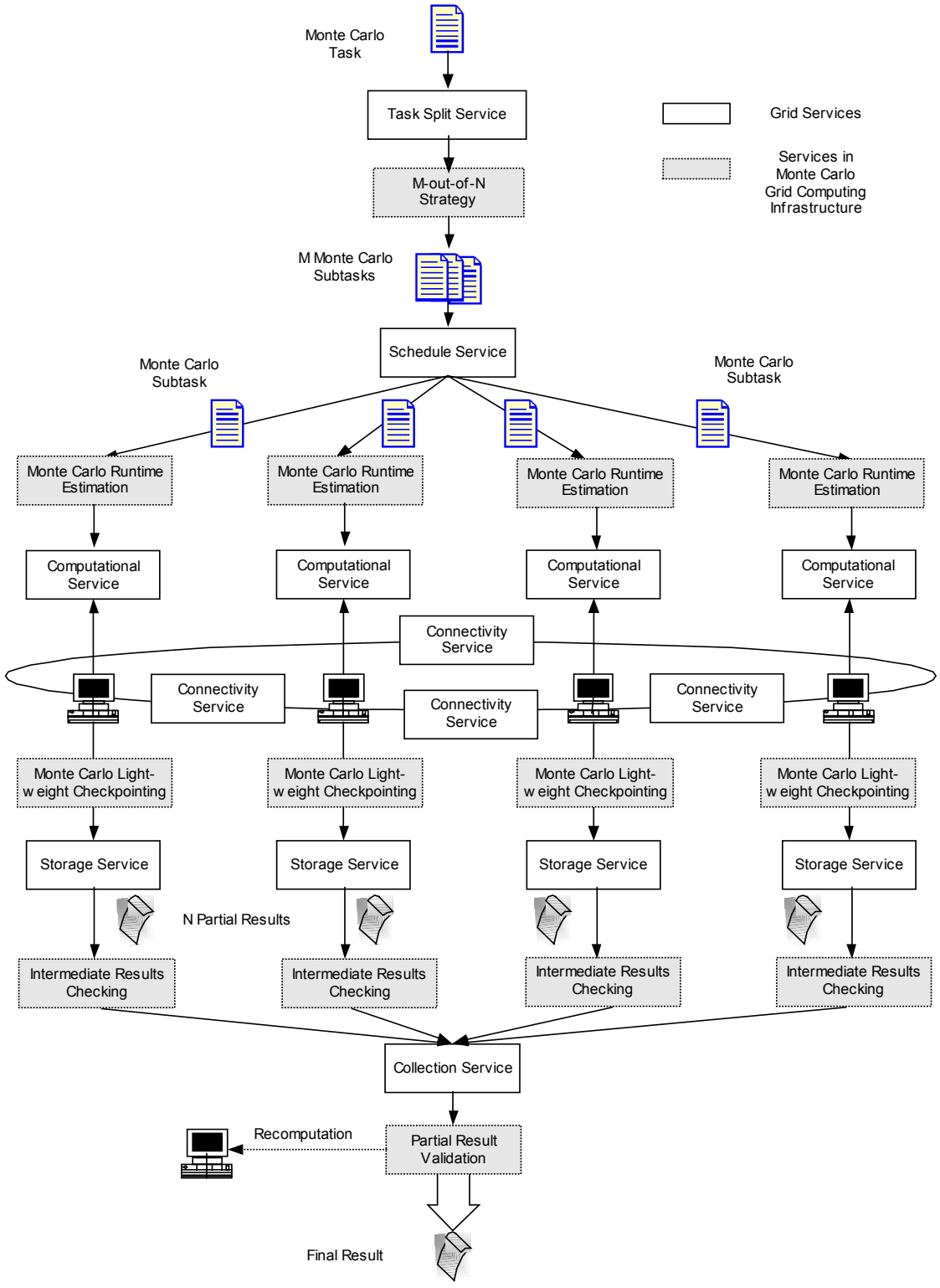

Figure 4.2 The Diagram of Grid-based Monte Carlo Computations

4.4 GCIMCA Working Paradigm

4.4.1 Working Paradigm Overview

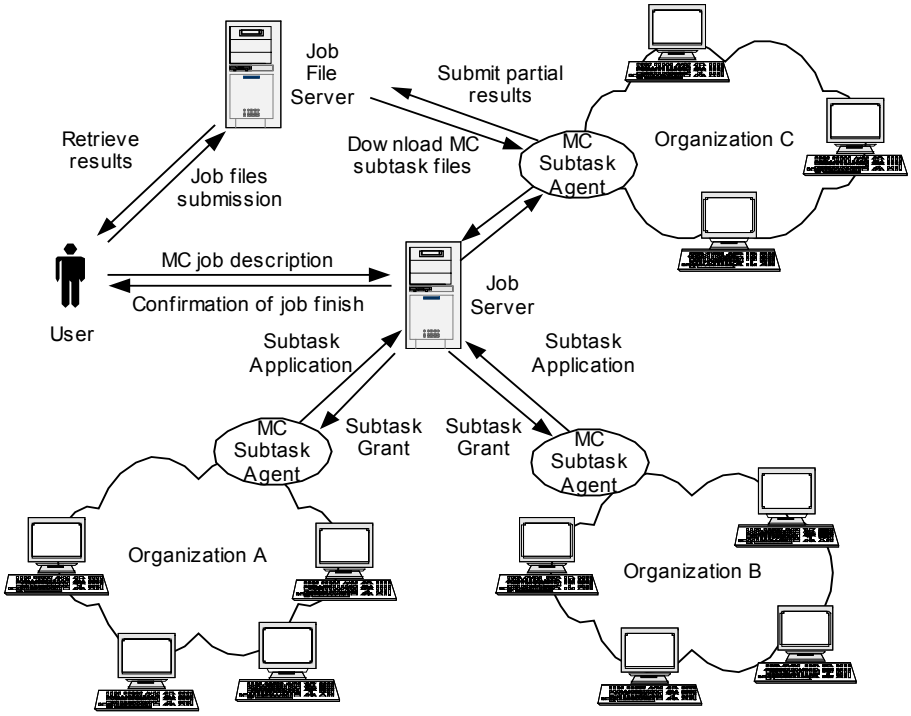

Figure 4.3 The Working Paradigm for GCIMCA

Figure 4.3 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.

4.4.2 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 4.4 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 4.4 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 [58] 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.

4.4.3 Passive-Mode Subtask Scheduling

Unlike the design of most existing distributed and parallel computing systems, such as Condor [19], Javelin [32], Charlotte [31] and HARNESS [64], 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 4.5 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.

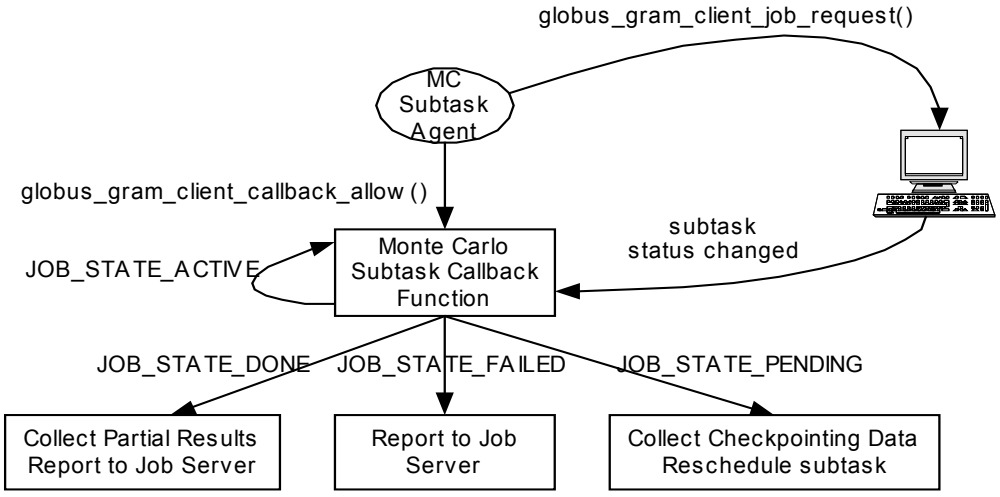

Figure 4.5 Remote Execution of a Monte Carlo Subtask

4.5 Implementation of GCIMCA Services

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

The main idea of the *N-out-of-M* strategy 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 more reliable 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 streams, 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.

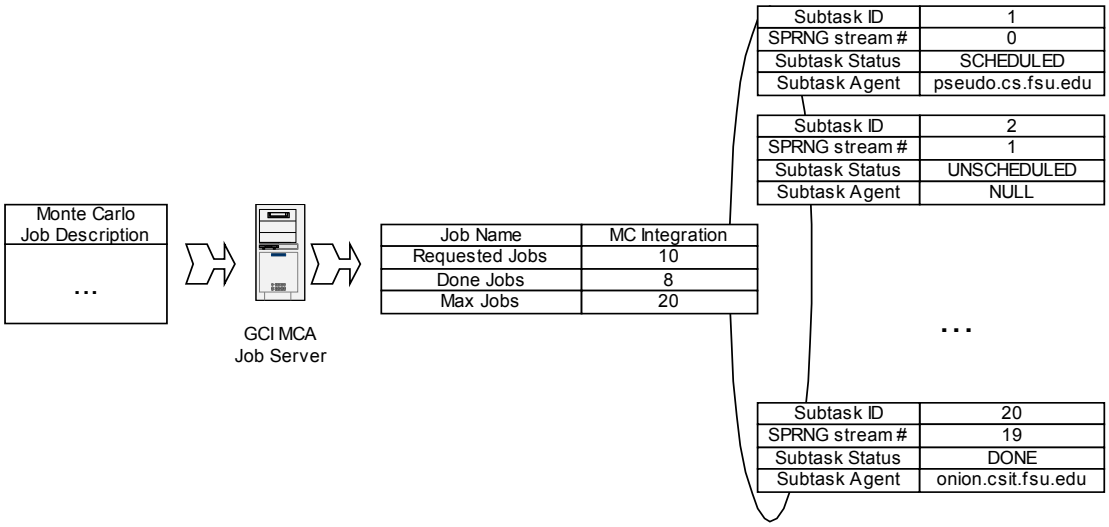

Figure 4.6 Implementation of the *N-out-of-M* Scheduling Strategy in GCIMCA

Figure 4.6 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.

4.5.2 Monte Carlo Lightweight Checkpointing

A long-running computational task on a grid node must be prepared for node unavailability. Compared to process-level checkpointing [20], 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. 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. GCIMCA takes advantage of this programming structure of Monte Carlo applications to implement the 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 [63] 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 4.7 shows the flowchart of GCIMCA's implementation of Monte Carlo application-level checkpointing and recovery.

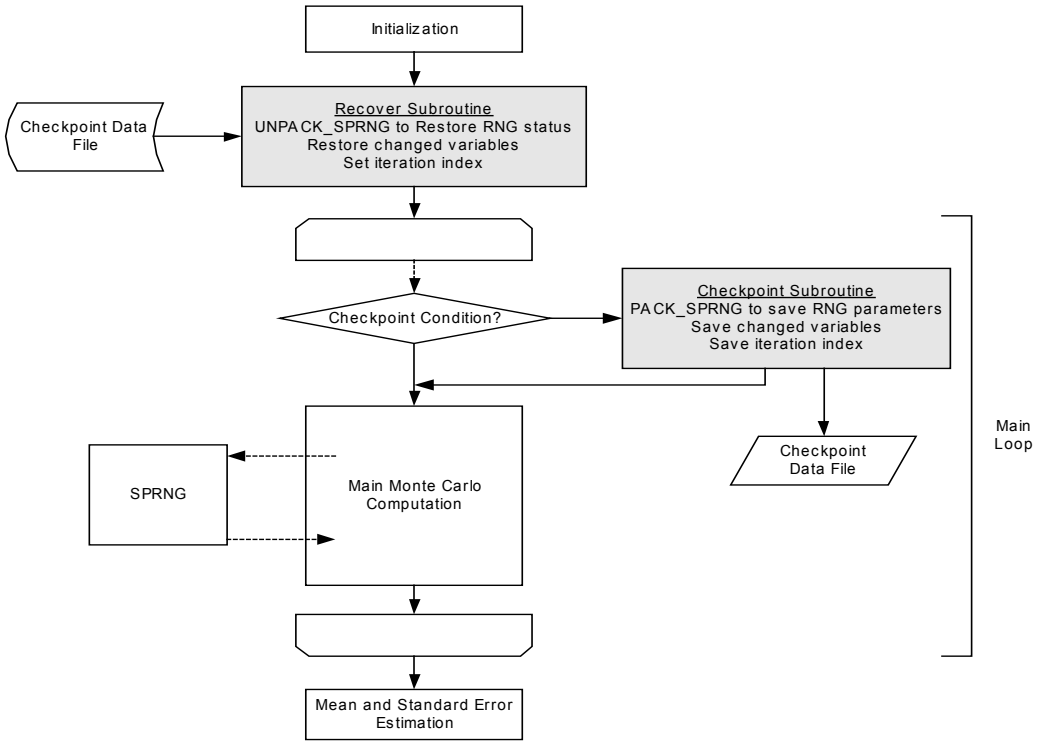

Figure 4.7 GCIMCA Implementation of Monte Carlo Application-Level Checkpointing

4.5.3 Partial Result Validation and Intermediate Value Checking

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. 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 determine whether the subtask is actually faithfully executed. The underlying pseudorandom numbers in Monte Carlo applications are the perfect candidates to use as the intermediate values. The intermediate value checking service in GCIMCA uses a simple strategy to validate a result from subtasks by tracing certain predetermined random numbers in 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 Nth, $2N$th, …, kNth 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 Nth, $2N$th, …, kNth random numbers applying the specific generator in the SPRNG library with the same seed and equivalent parameters as used in this subtask. A mismatch indicates problems during the execution of the subtask.

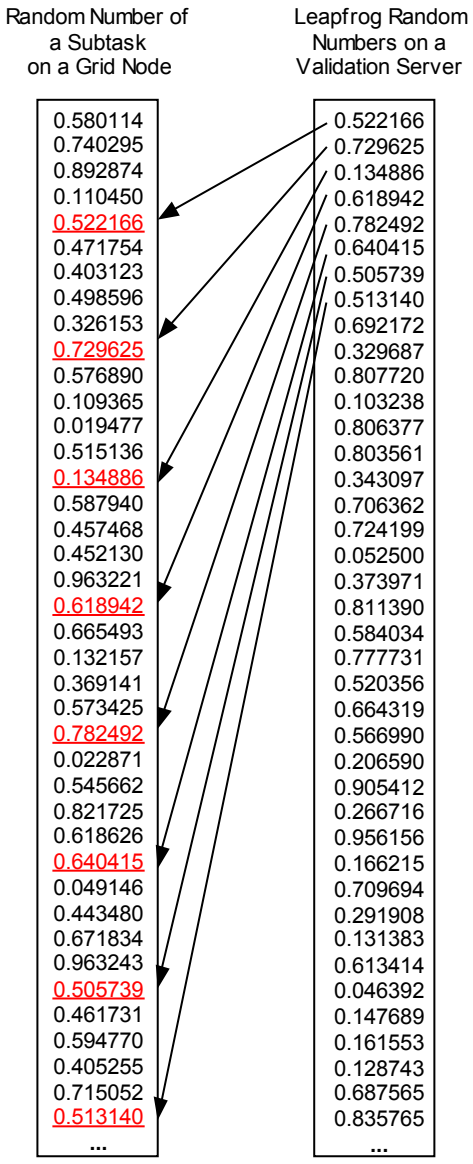

Figure 4.8 Match of the Random Numbers in a Subtask and the Ones Regenerated by Leapfrog on the Job Server

To regenerate the *N*th, *2N*th, …, *kN*th random numbers of a specific generator, we implement a leapfrog function in the SPRNG library.

```
int leapfrog(int *igenptr, int leaplength);
```

The leapfrog function retrieves the parameters of the original generator, computes the new parameters of the leapfrog generator based on the leaping length, and finally

converts the original generator to the leapfrog generator. After calling the leapfrog function, we can get the leapfrog generator and economically generate the leaping random number sequence. Figure 4.8 shows the match of the random numbers used in a subtask and the ones regenerated by leapfrog in the job server.

The implementation algorithms of grid-based Monte Carlo partial result validation and checking are relatively simple. Figure 4.9 shows the algorithm flowchart for Monte Carlo partial result validation and intermediate value check. To check the intermediate values, the selected random numbers are reproduced on the job server by calling the SPRNG leapfrog function. When the partial result returns, the recorded random number values are retrieved. By simply comparing the random numbers in the returned partial result data with those reproduced by the server-side program, we will be able to determine whether this subtask is actually executed faithfully or not. The partial result validation process starts when the required number of partial results are ready. It will calculate the confidence interval based on the mean and the statistical standard error of the partial results. Then, in the next step, for each partial result, the partial result validation process will examine if it resides in the appropriate confidence interval. If the answer is yes, this partial result is regarded as an acceptable result; otherwise, the particular subtask that generates this partial result needs to be re-executed on a trustable service provider. Notice that in order to use the partial result validation and intermediate value checking facilities, in the Monte Carlo program, the Monte Carlo application developer has to explicitly specify the intermediate values and the quantitative values within the partial result data file that possess statistical property.

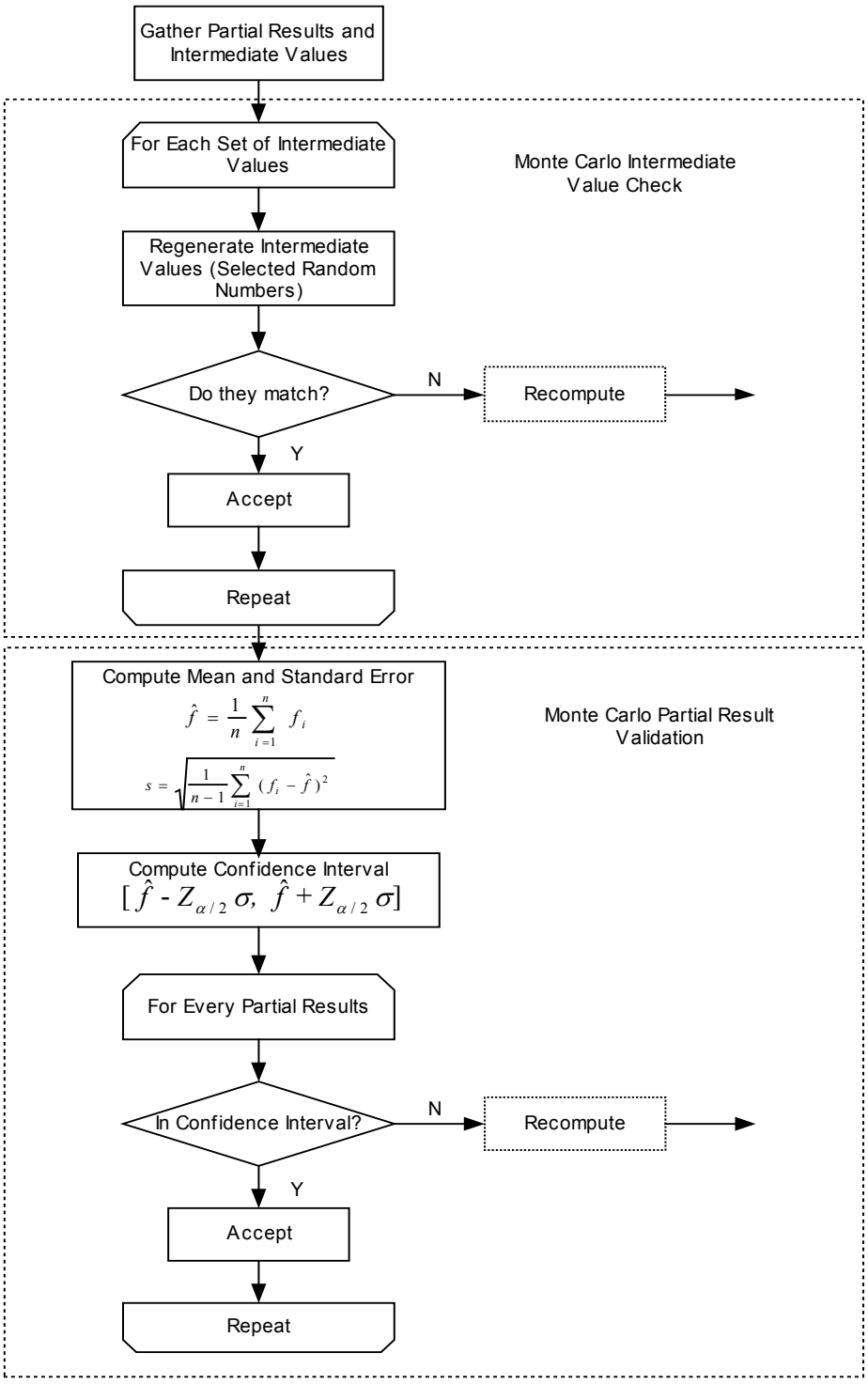

Figure 4.9 Flow Chart of Grid Monte Carlo Partial Result Validation and Intermediate Value Checking

CHAPTER 5

MONTE CARLO APPLICATIONS ON THE GRID

With our Monte Carlo grid-computing infrastructure, GCIMCA, many Monte Carlo applications can be run on it to obtain better performance, accuracy, and trustworthiness. In this chapter, we will choose a fundamental Monte Carlo application – distributed Monte Carlo integration – as a "simple warm-up problem" in Section 5.1. Then, in Section 5.2, we will use a real-life engineering application – large-scale Molecular Dynamics/Brownian Dynamics simulation for Ligand-Receptor interaction in structured protein systems [81], to demonstrate the utilization and power of the Monte Carlo grid-computing infrastructure. Experiments are done on the computational grid to compare with conventional distributed/parallel computing environments.

5.1 Grid-based Multidimensional Monte Carlo Integration

The Monte Carlo method seems to be the only feasible computational method to evaluate complicated integrals in high dimension. Our grid-based Monte Carlo integration application has the functionality of evaluating multi-dimensional integrals, avoiding "the curse of dimensionality." We take advantage of the services and tools in GCIMCA for Monte Carlo applications to demonstrate its utilization.

5.1.1 Introduction to Monte Carlo Multidimensional Integration

Suppose we need to calculate the *s*-dimensional integral

$$\theta = \int_0^1 \cdots \int_0^1 f(x_1, \ldots, x_s)\, dx_1 \cdots dx_s \tag{21}$$

using crude Monte Carlo. If $\xi_1, \ldots, \xi_n$ are independent uniformly distributed random number s-dimensional vectors, then the quantities,

$$f_i = f(\xi_i), \tag{22}$$

are independent random variates with expectation θ and variance

$$\sigma^2 = \int_0^1 \cdots \int_0^1 (f(x_1, \ldots, x_s) - \theta)^2\, dx_1 \ldots dx_s . \tag{23}$$

Therefore,

$$\bar{f} = \frac{1}{n} \sum_{i=1}^{n} f_i , \tag{24}$$

is an unbiased estimator of θ with standard error $\frac{\sigma}{\sqrt{n}}$ [39]. Thus, we estimate θ by computing $\bar{f}$ that will have mean θ. In practice, we probably don't know the standard error. Instead, we estimate the variance from the formula,

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (f_i - \bar{f})^2 = \sum_{i=1}^{n} f_i^2 - \bar{f}^2 , \tag{25}$$

to provide a confidence interval for our estimate.

In conventional numerical methods for evaluating s-dimensional integrals, with increasing dimension, s, the error bound, $O(N^{-\alpha/s})$, increases drastically; hence the required number of integration nodes increases exponentially – this is called the "curse of dimensionality." With crude Monte Carlo, the quantity to be calculated is interpreted via a stochastic model and subsequently estimated by random sampling. The probabilistic convergence rate of crude Monte Carlo method is known to be approximately $O(N^{-1/2})$, which is independent of dimension s, and thus avoids the "curse of dimensionality." [11]

5.1.2 Experiencing Monte Carlo Multidimensional Integration on a Computational Grid

We implemented a multidimensional Monte Carlo integration program using crude Monte Carlo. The program can evaluate the integral (13) described in Example 3.1 of Chapter 3 based on GCIMCA. Then, we used the Monte Carlo job description file in Figure 4.4 to submit a task to implement a *10-out-of-40* scheduling. Here, each subtask is based on the evaluation of a unique random number stream generated by the SPRNG library. To obtain more grid nodes, we also enabled the Subtask Agent to schedule a subtask in a Condor pool using the following Globus Resource Specification Language (RSL) to specify the subtask description of the Monte Carlo integration program:

"(arguments=-newrun)(count=1)
(executable=mcint.exe)(stdin=mcint.in)(stdout=mcint.out)(stderr=mcint.err)
(jobType=condor)"

The subtask description in RSL gives the application data, application location, runtime parameters, and the target execution platform -- a Condor pool. Table 5.1 (below) tabulates the computational results and wallclock time of 10 runs on a computational grid using the *10-out-of-40* scheduling and *10-out-of-10* scheduling. At each run, 40 and 10 subtasks were actually carried out using the *10-out-of-40* scheduling and *10-out-of-10* scheduling, respectively, with each subtask evaluating 10^9 random samples. When 10 subtasks were complete, we recorded the wallclock time and calculated the corresponding mean and standard deviation of each run.

Table 5.1 Comparison of the *N-out-of-M* and *N-out-of-N* Strategy in Grid-based Monte Carlo Multidimensional Integration

# of Run	10-out-of-40			10-out-of-10		
	Wallclock Time (s)	Mean	Standard Deviation	Wallclock Time (s)	Mean	Standard Deviation
1	1001	103.788513	0.028695	10793	103.817396	0.028974
2	875	103.808234	0.028661	967		
3	1534	103.818967	0.028735	8695		
4	955	103.830010	0.028711	12436		
5	1678	103.860001	0.028910	5246		
6	2103	103.852748	0.028709	2890		
7	1274	103.866856	0.028866	1745		
8	962	103.822717	0.028736	16780		
9	1524	103.828391	0.028683	6489		
10	1320	103.764991	0.028633	6340		

From Table 5.1, we see that the *10-out-of-10* scheduling has task completion times varying in a wide range, while *10-out-of-40* scheduling has relatively more predictable. Also, the *10-out-of-40* scheduling has much shorter average task completion time (1323s) than that of the *10-out-of-10* scheduling (6238s). Since in both schedules, the Monte Carlo integration tasks evaluated the same number of random samples, these computations have very similar accuracy.

The application-level lightweight checkpointing in Monte Carlo integration is rather simple -- only the current iteration index, the status of the random number generator, the accumulated sum of the evaluation on previous random samples, and the current variance must be stored. The Subtask Agent can then use the following RSL to resume the subtask on another grid node.

"(arguments=-checkpoint)(count=1)

(executable=mcint.exe)(stdin=checkpoint.in)

(stdout=mcint.out)(stderr=mcint.err)"

We validate the partial results of the grid-based Monte Carlo integration subtasks we obtained. Figure 5.1 shows the distribution of 100 generated partial results: 63 partial results are located within 1 standard deviation of the mean, 93 partial results within 2 standard deviations, 98 of the 100 partial results within 3 standard deviations, and all within 4 standard deviations. A partial result fallen out of the confidence interval with 4 standard deviations can be easily detected as suspect and either recomputed on a trusted host or discarded.

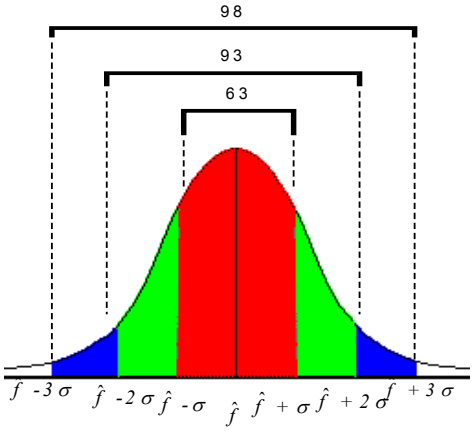

Figure 5.1 Partial Result Distribution of Monte Carlo Integration Subtasks

In grid-based Monte Carlo integration, we also examine random numbers as intermediate values to help validate if the subtask is faithfully executed. In this experiment, we used the LCG generator in the SPRNG library. Here we recorded the value of the first random number of every block of 10^7 random numbers. Table 5.2 shows the match of the first several random numbers in one of these subtasks. We notice that the generation of 10^9 pseudorandom numbers costs 222 seconds on a Linux system with a Pentium IV 1.6GHz processor; nevertheless, using the leapfrog technique to regenerate 100 random numbers with 10^7 as leaping distance requires less than 1 second on the same computer.

Table 5.2 Matches of Random Number as Intermediate Values and the Regenerated Random Numbers

Random Number Recorded in a Subtask	Random Number Regenerated
0.575170	0.575170
0.818598	0.818598
0.819430	0.819430
0.865870	0.865870
0.647430	0.647430
0.286175	0.286175
…	…

5.1.3 Summary of Grid-based Monte Carlo Integration

The Monte Carlo multidimensional integration program is a typical and representative example to demonstrate running a distributed Monte Carlo application on the grid. "Every Monte Carlo computation that leads to quantitative results may be regarded as estimating the value of a multiple integral." [11] Monte Carlo integration constitutes the basis for many other Monte Carlo calculations. For example, using Monte Carlo methods to solve Boundary Value Problems (BVP) may be regarded as estimating the value of a multiple integral. This grid-based multidimensional Monte Carlo integration example shows the way to use the services and development toolkits in GCIMCA. It seems clear that the techniques in GCIMCA can be applied to other Monte Carlo applications based on integral evaluation as well. However, we plan to investigate these application classes in detail after the dissertation.

5.2 Grid-based Molecular Dynamics (MD)/Brownian Dynamics (BD) Simulation

In the hybrid Molecular Dynamics (MD)/Brownian Dynamics (BD) algorithm for simulating the long-time, nonequilibrium dynamics of receptor-ligand interactions, the evaluation of the force autocorrelation function can be computationally costly but fortunately is highly amenable to multimode processing methods. In this chapter, taking advantage of the computational grid's large-scale computational resources and the nice characteristics of grid-based Monte Carlo applications, we developed a grid-based receptor-ligand interactions simulation application using the MD/BD algorithm and the facilities of GCIMCA. We expect to provide high-performance and trustworthy computing for analyzing long-time dynamics of proteins and protein-protein interaction to predict and understand cell signaling processes and small molecule drug efficacies. Our preliminary results showed that our grid-based application could provide a faster and more accurate computation of the force autocorrelation function in our MD/BD simulation than previous parallel implementations [81].

5.2.1 MD/BD Simulation

Prediction of the long-time, nonequilibrium dynamics of receptor-ligand interactions for structured proteins in a host fluid is of critical importance to the understanding of infectious diseases, immunology, the development of "target" drugs, and biological separations. However, such processes take place on time scales on the order of milliseconds to seconds, which prevents "brute-force" real-time molecular or atomic simulations from determining the absolute ligand binding rates to receptor targets. In a previous study [82], we implemented a hybrid Molecular Dynamics (MD)/Brownian Dynamics (BD) algorithm which utilizes the underlying, disparate time scales involved and overcomes the limitations of brute-force approaches. Single and isolated proteins, protein with charge effects, and D-peptide/HIV capsid protein systems were investigated using the hybrid MD/BD algorithm [82].

Within the hybrid MD/BD algorithm, the calculation of the force autocorrelation function to generate the grand particle friction tensor forms the basis of the most computationally costly part, which requires large amounts of CPU times. Even on an advanced supercomputer, this computation takes from days to months, and thus becomes the performance bottleneck of the MD/BD simulation. Fortunately, this part of the computation uses Monte Carlo methods[2], which are computationally intensive but naturally parallel. It is very amenable to the emerging grid-computing environment, characterized by "large-scale sharing and cooperation of dynamically distributed resources, such as CPU cycles, communication bandwidth, and data, to constitute a computational environment" [30]. A large-scale computational grid can, in principle, offer a tremendous amount of low-cost computational power, which attracts us to utilize the computational grid for our MD/BD application. On the other hand, our previous studies in grid-based Monte Carlo applications in Chapter 3 and 4 showed that Monte Carlo's statistical nature could be applied to improve the performance and enforce the trustworthiness of grid computing at the application level. The rest of this section will study the development of the grid-based nonequilibrium, multiple-time scale simulation application of ligand-receptor interactions. We take advantage of the services of a computational grid and the characteristics of grid-based Monte Carlo applications to provide high-performance and trustworthy computation for predicting and understanding the dynamics of structured protein systems.

5.2.2 Hybrid MD/BD Algorithm

5.2.2.1 Introduction to Hybrid MD/BD Algorithm

[2] The hybrid MD/BD algorithm does not use the Monte Carlo method based on the Metropolis method; hence, it is not a typical Monte Carlo molecular modeling application. Nevertheless, since the Monte Carlo method is used to generate the configuration of host liquid in the hybrid MD/BD algorithm, the MD/BD simulation has the characteristics of generic Monte Carlo applications.

In a previous study [83] of the behavior of the many-bodied friction tensor for particles immersed in a rarefied, "free-molecule" gas, a molecular dynamics method was used. It was noted that the molecular dynamics method could be used to study the long-time behavior of Brownian particles by a two-step procedure. This procedure is illustrated as following:

1) For a given particle configuration, the many-body friction tensor is determined from MD through the analysis of the force autocorrelation function. In this step, the particle coordinates are kept fixed according to a fluctuation-dissipation type relation that gives the (time-independent) friction tensor in terms of the force autocorrelation function.
2) The Fokker-Planck (FP) equation, which describes the dynamics of a single structured Brownian particle in a molecular fluid, is solved for discrete times assuming that the friction tensor remains constant over the time step. The particles are advanced to new positions according to the integrated FP equation.

Simply speaking, the key to the hybrid MD/BD numerical algorithm is that the host fluid relaxes to an equilibrium state in the potential field of the Brownian particle over very short times, on the order of picoseconds or less. However, significant changes in the Brownian particles's position and orientation take place on much longer time scales, on the order of microseconds or longer. This allows us to set the particle diffusion properties using MD and time correlation analysis (the picosecond calculation), and then "leap ahead" in time moving the particle according to a simple 6-dimensional (3 positional and 3 orientational coordinates) BD time step. The entire process, MD followed by BD is repeated and MD is only performed at the beginning of each BD time step (the microsecond calculation). This MD/BD algorithm is based on a multiple time scale analysis of the total system Hamiltonian, including all atomic molecular structure information for the system: water, ligand, and receptor. The results allow the study of the long-time dynamics of macromolecules in complex systems where complete molecular details of the macromolecule, surface, and solvent can be incorporated. The theoretical

background and a detailed review of the hybrid MD/BD algorithm can be found in [81, 82, 83, 84, 85, 86].

5.2.2.2 Hybrid MD/BD Algorithm Implementation

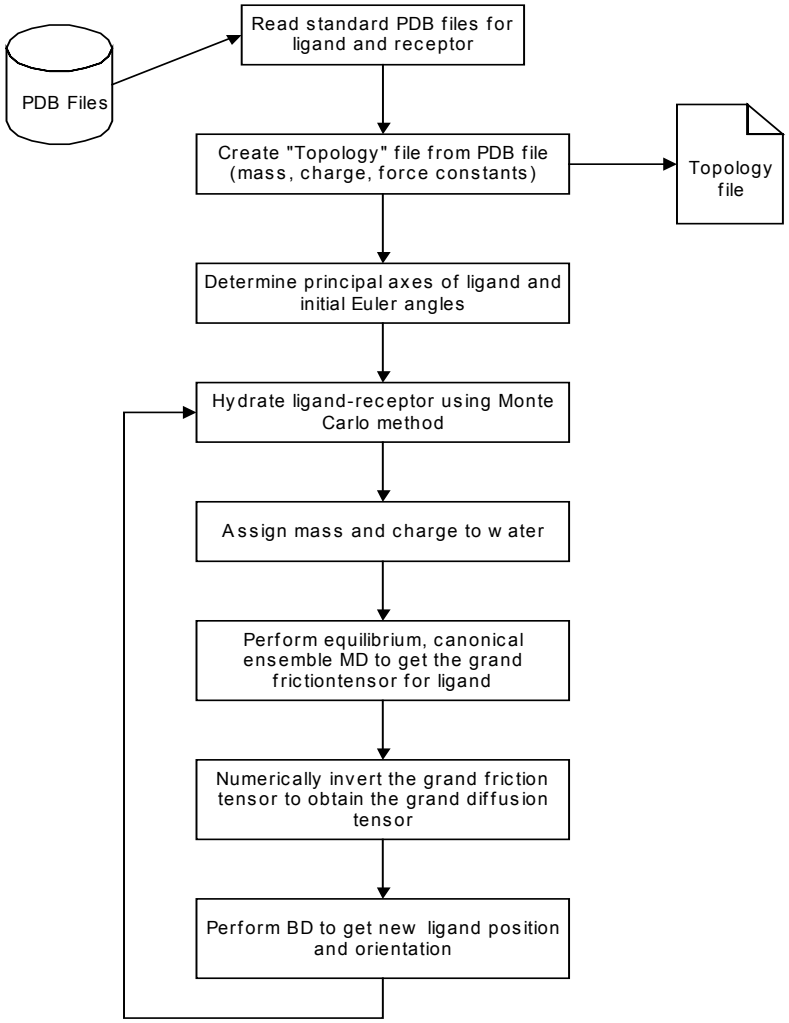

Figure 5.2: Flowchart of the Hybrid MD/BD Algorithm

The hybrid MD/BD algorithm was implemented in [87] to study the D-peptide/HIV system. The general computational MD/BD algorithm is shown in Figure 5.2 [81]. The computational scheme begins by reading a standard PDB file from the

Protein Data Bank [88] for both ligand and receptor. This file is then converted to a "topology" file that includes computationally critical information on atomic mass, residue charge, and Lennard-Jones interaction force constants. Next, the ligand and receptor must be hydrated using SOLVATE [89], which uses a Monte Carlo method. For the molecular model of water, the so-called modified Simple Point Charge (SPC) model [90] is used with long-range electrostatic inter-atomic interactions accounted for by a modified Poisson-Bolzmann reaction field method, which uses an acceptance-rejection Monte Carlo approach. The center of mass and body fixed axes along the principal axes of inertia for the ligand are initially computed. This sets the body-fixed coordinates and initial Euler angles, the latter of which give the orientation of the body relative to the space fixed frame. MD is then used to determine the particle grand friction tensor. The grand friction tensor is numerically inverted to obtain the grand diffusion tensor. The grand diffusion tensor is then utilized to perform the BD move on a time step of around 10^{-5} seconds. The macromolecule position and orientation change by only a couple of percent or less over this time period. The new atomic positions are updated based on the BD move and the entire process, MD followed by BD, is repeated.

5.2.2.3 The Force Autocorrelation Function

The force autocorrelation function can be obtained by conducting standard canonical, equilibrium molecular dynamics simulations. The particle is considered to be composed of a large number of molecules each interacting with the fluid molecules according to a Lennard-Jones potential [85]. Suppose the Brownian particle is composed of M molecules and the fluid consists of N molecules. The computation of the force autocorrelation function is $O(N^2 + M*N)$ at every time step, which is very computational costly.

A confidence interval in the autocorrelation values, CI, is obtained from the Tchebycheff inequality as

$$CI = 1 - \frac{\sigma^2}{n_r \varepsilon^2}, \tag{21}$$

where ε is the error in the autocorrelation, σ^2 is the variance, and n_r is the number of repetitions (ensembles). The error is proportional to the reciprocal square root of the number of repetitions n_r, i.e.,

$$\varepsilon \sim 1/n_r^{1/2}. \tag{22}$$

Thus with increasing number of repetitions, the error in the autocorrelation is reduced. Our results show that at least 20 ensembles are minimally necessary, and more would be desirable for more accurate results.

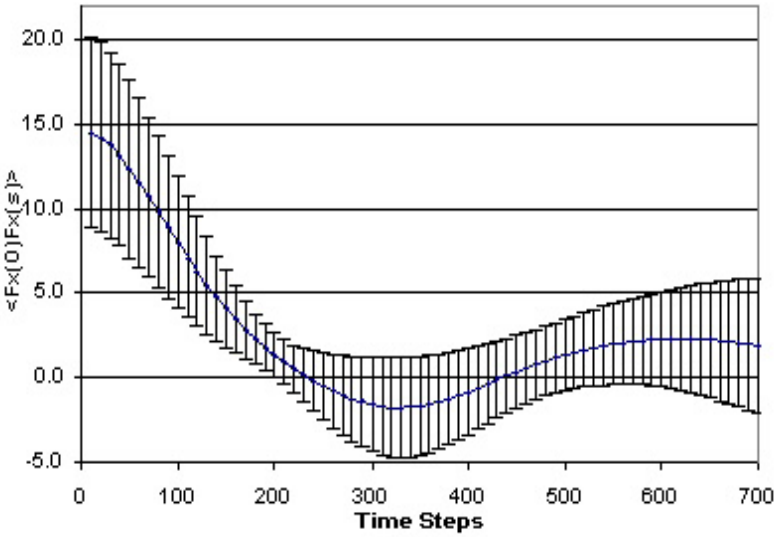

Figure 5.3: The xx-component of the Force Autocorrelation Function in One Standard Deviation for a D-peptide

Figure 5.3 shows the xx-component of the force autocorrelation function for a D-peptide within 1 standard deviation with 40 ensembles. Our experiment of the computation in Figure 5.3 was implemented using MPI. The computation simulates the D-peptide with 372 molecules in the system with around 10,000 water molecules running in 60,000 time steps. The experiment took almost 8 days on a DEC Alpha DS10 6/466

with 256 MB of DRAM with 4 processors for a single step of MD simulation, and more than a month on a serial DEC Alpha DS10. We can expect greater time consumption for a particle with more molecules, or a system with more host fluid molecules. The high computational cost of evaluating the force autocorrelation function constrains from performing more steps of the MD/BD simulation to study the behavior of particles and particle interactions. More importantly, since the friction tensor in the BD is the integral of the force autocorrelation, the inaccuracy in the MD may mislead the computation of the BD. This error can even propagate further in the MD/BD simulation.

Deeper study of the MD part of the algorithm shows that the force autocorrelation function is particularly amenable to multiprocessor systems [81]. In parallel MD simulation, each node can represent one member (3,000 time steps) of the ensemble allowing hundreds and thousands of ensembles to be included. More importantly, once scheduled, each ensemble's computation is based on its own fluid configuration, which is independent with no intercommunication needed. Also, each independent execution time costs a few hours or less depending on processor speed. This property of the autocorrelation computation motivates us to take advantage of the tremendously large and low-cost computational power in a computational grid for our MD/BD dynamics simulation for this structured protein system.

5.2.3 Implementation of Grid-based MD/BD Simulation

5.2.3.1 Grid-based MD/BD Simulation Application Overview

To develop a grid-based hybrid MD/BD simulation application, we need to utilize certain grid services. First of all, the task split service is used to define the data set and initial conditions for each ensemble computation, e.g., the ligand and receptor configurations, the host fluid configuration, the Lennard-Jones constants, and the parameters for random number stream. Each ensemble data and program are packed into a grid subtask. Secondly, the task schedule service is used to distribute these subtasks to

individual computational service providers. During the execution of the subtask, the storage service is used to store the checkpointing data, intermediate results, and subtask results. Thirdly, when the partial results are ready, the collection service is responsible for gathering them all and validating each partial result. Finally, based on the computational results of all ensembles, we can assemble the estimate of the force autocorrelation function and estimate its statistical error. After the MD simulation using the grid environment, the BD simulation follows and updates the new atomic positions in the particle. The above process can be repeated for the next BD moves.

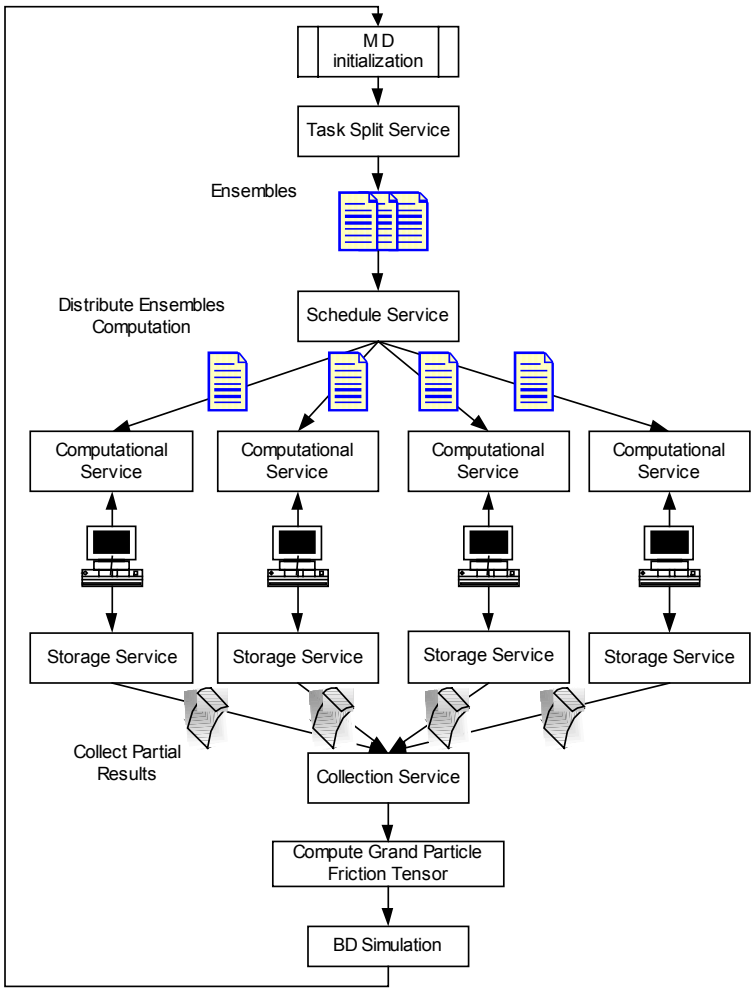

Figure 5.4: Working Paradigm for the Grid-based MD/BD Simulation on a Structured Protein System

Figure 5.4 shows the working paradigm of the grid-based hybrid MD/BD simulation application. Furthermore, the MD part of the computation is a typical grid-based Monte Carlo computation, which exhibits characteristics that can be used to improve the application's performance and trustworthiness in the grid. We can take advantage of these properties to optimize the MD computation.

5.2.3.2 Results of N-out-of-M Scheduling

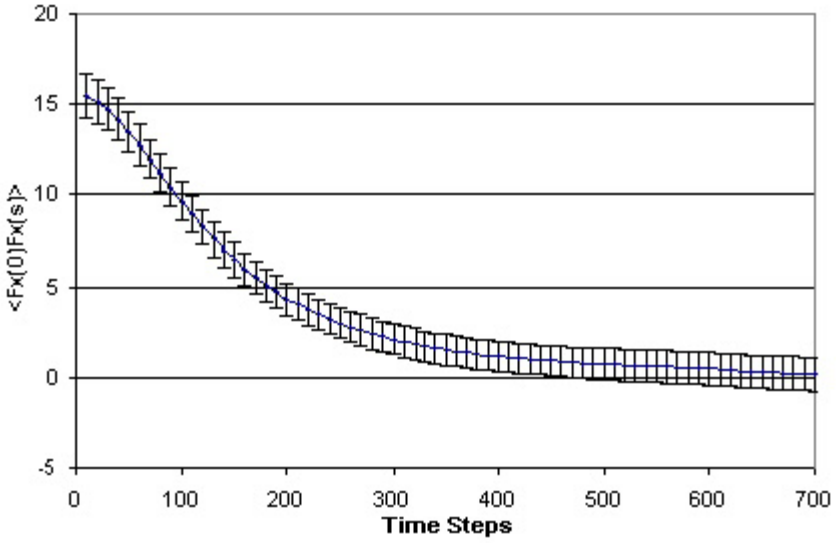

Figure 5.5: the xx-component of the force autocorrelation function in one standard deviation for a D-peptide on Condor with 400 ensembles

We apply our *N-out-of-M* scheduling technique to schedule the MD computation subtasks to the computational grid. Here, we performed our experiments using GCIMCA's *N-out-of-M* scheduling service. Some nodes in a Condor pool also participated in the computation. Figure 5.5 shows our preliminary results of grid-based MD/BD simulation using GCondor with 40 nodes each carrying the computation of 10 ensembles. The SPRNG library is used to provide parallel random number streams. The D-peptide data in this experiment is the same as the one in Figure 4.3. Using the *N-out-*

of-M subtasks scheduling strategy, this computation took around 8 days to obtain 400 ensembles computational results of the force autocorrelation function (by scattering the 400 ensembles to more nodes in the grid, we can expect an even shorter completion time). We can find that the xx-component of the force autocorrelation function has a much smaller error bound than the one in Figure 5.3. More specifically, Table 5.3 shows the statistical error bound estimates of xx-component of the force autocorrelation function with 90% confidence in this experiment. We notice that statistical error decreases while increasing the number of ensembles. The computations of other components of the force autocorrelation function exhibit similar behavior.

Table 5.3: The Error of xx-component of the Force Autocorrelation Function with 90% Confidence. The Error Decreases with Increasing Number of Ensembles.

# of Ensembles	Std Deviation	Error Bound
10	4.10	12.97
100	1.39	4.40
200	0.943	2.97
400	0.671	2.11

5.2.3.3 Checkpointing the MD Simulation

The execution of each MD subtask takes a relative long time on a grid node, and so the implementation of checkpointing is quite necessary. We implemented the application-level lightweight checkpoint in our MD simulation program. Since at each time step in the MD simulation, the positions of the atoms in the structured proteins remain the same, the only data that change are the configuration of the host fluid, such as the atoms' locations and velocities. Thus, the checkpointing data that the subroutine needs to save are the configuration of the host fluid, the current time step, and the force autocorrelation function values from the previous time steps. The checkpointing data are

stored into the checkpointing file. Based on this checkpointing data, the recover subroutine can easily restore an interrupted MD calculation.

5.2.3.4 Partial Result Validation

In Chapter 3, we discussed a partial result validation method for point solutions taking advantage of the statistical nature of Monte Carlo applications. This method can be easily extended and used in our grid-based MD/BD simulation application to validate the force autocorrelation function curve from each ensemble. Based on the force autocorrelation function values at every time step, we calculate its mean, standard deviation, and then the corresponding confidence interval. The upper bound endpoints of all these confidence intervals at different time steps provide an upper bound curve, and the lower bound endpoints provide a lower bound. If a force autocorrelation function curve from an ensemble lies in the area between the upper bound and lower bound curves, we consider the partial result of this ensemble computation as being trustworthy; otherwise, it is suspect, and we may need to rerun this particular subtask for further verification.

Figure 5.6 shows the partial result validation mechanism in our grid-based MD/BD simulation. We find that all of the force autocorrelation function curves obtained from different nodes in the computational grid lie in the area between the upper bound and lower bound curves within 3 standard deviation of the mean. We thus regard all of them as trustworthy computations.

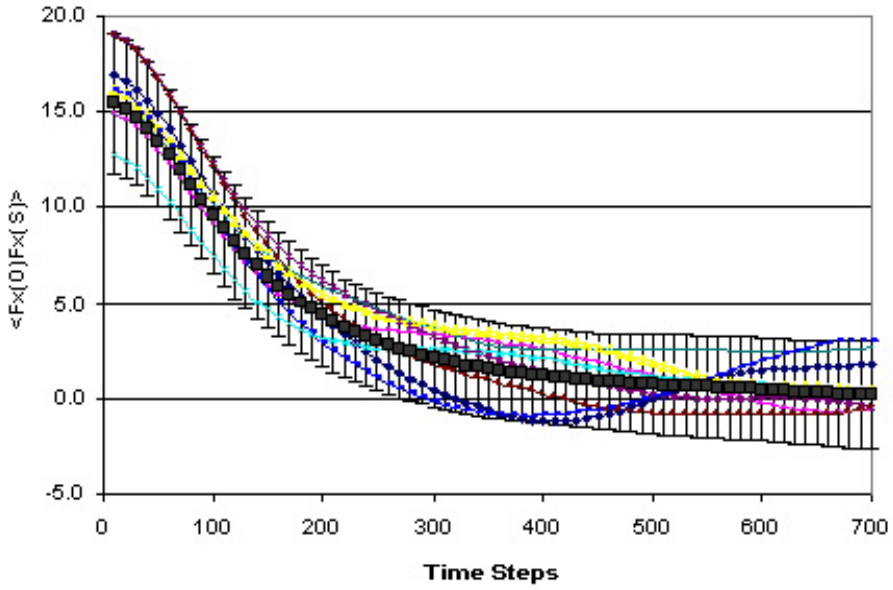

Figure 5.6: Partial Result Validation of xx-component of the Force Autocorrelation Function. Each Curve Lies in the Error Bounds of 3σ.

CHAPTER 6

TECHNIQUES EXTENSION

In previous chapters, we analyzed the inherent characteristics of Monte Carlo applications, developed the techniques for high-performance and trustworthy Monte Carlo computations on the grid, implemented the grid-computing infrastructure, GCIMCA, based on these techniques, and applied Monte Carlo applications to the grid. All these approaches are Monte Carlo application oriented. In further study, we find that some of these techniques may also be adaptable to other grid-computing applications. In this chapter, we try to explore the extension of these techniques to other grid-computing applications. First, in Section 6.1, we analyze the grid-based quasi-Monte Carlo applications, which are closely related to Monte Carlo applications but use quasirandom samples instead. Then, in Section 6.2 and 6.3, we discuss the techniques of intermediate value checking based on pseudorandom numbers and application-level lightweight checkpointing for more generic grid-based applications, respectively.

6.1 Grid-based Quasi-Monte Carlo Applications

6.1.1 The *N-out-of-M* Subtask Schedule for Grid-based Quasi-Monte Carlo Applications

In grid-based Monte Carlo applications, the *N-out-of-M* subtask schedule strategy requires the 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. 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 the order of N^{-1}, then the sequence S that is combined from S_1 and S_2 will have discrepancy of the order of $(2N)^{-1}$. Unfortunately, this is not true. The simplest counterexample is the combination of two identical low-discrepancy sequences 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 on parallel quasirandom number sequences. The Koksma-Hlawka inequality [52] is the foundation of analyzing quasi-Monte Carlo integration error. Based on the Koksma-Hlawka inequality, we deduce a Lemma (6.2) that provides an upper bound on the error for a parallel quasi-Monte Carlo integration using multiple low-discrepancy sequences.

Theorem 6.1 (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,

$$\varepsilon[f] \le V[f] D_N^* . \tag{23}$$

Here $V[f]$ is the total variation of f, in the sense of Hardy-Krause, D_N^* [3] is the star discrepancy of sequence $X = \{x_0, \ldots, x_{N-1}\}$, and $\varepsilon[f]$ is defined as

$$\varepsilon[f] = \int_{I^d} f(x)dx - \frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{24}$$

where d is the dimension of f.

---

[3] For a sequence of N points $X = \{x_0, \ldots, x_{N-1}\}$ in the d-dimensional unit cube I^d. For any box J with one corner at the origin in I^d, the star discrepancy D_N^* is defined as

$$D_N^* = \sup_{J \in I^d} \left| \mu_x(J) - \mu(J) \right|,$$

where $\mu_X(J) = \dfrac{\#\, of\ points\ in\ J}{N}$ is the discrete measure of J, i.e., the fraction of points of X in J, and $\mu(J)$ is the Euclidean measure of J, i.e., the volume of J.

Lemma 6.2 For M sequences $X_1 = \{x_{0,1}, \ldots, x_{N-1,1}\}$, $X_2 = \{x_{0,2}, \ldots, x_{N-1,2}\}$, $\ldots$, $X_M = \{x_{0,M}, \ldots, x_{N-1,M}\}$ 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,

$$\varepsilon[f] \le \frac{V[f]}{M} \sum_{i=1}^{M} D^*_{N,i} \,. \tag{25}$$

Proof: According to the Koksoma-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 6.2 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 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. Nevertheless, empirical experiments in Figure 6.1 show that the *N-out-of-M* strategy in quasi-Monte Carlo integration is rather effective. In these experiments, we divided a Sobol' sequence into 4 consecutive blocks, each block having an equal number of quasirandom numbers. Then, we evaluated the integral (13) in Chapter 3 using all combinations of any two of these blocks. In Figure 6.1, we see that the integral evaluations based on these combinations share a similar convergence rate. For our future

research of the *N-out-of-M* strategy for grid-based quasi-Monte Carlo applications, we may use the scrambling techniques for generating the parallel quasirandom number sequences with low discrepancies [91].

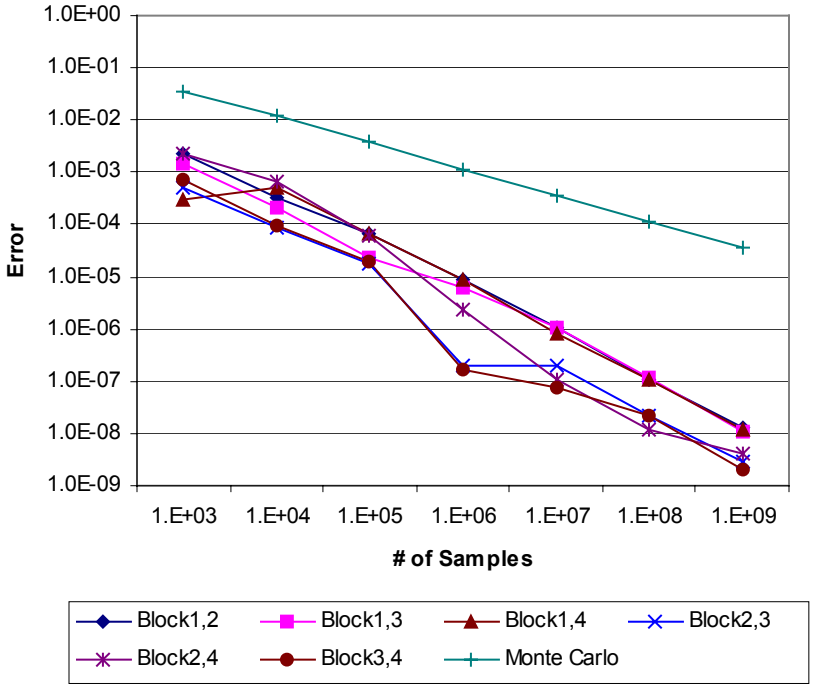

Figure 6.1: Convergence Analysis of Quasi-Monte Carlo using Different Blocks Combinations in a Quasirandom Number Sequence

6.1.2 Partial Result Validation for Grid-based Quasi-Monte Carlo Applications

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 so 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 that partial results from quasi-Monte Carlo subtasks are normally distributed. The partial validation method based

on all partial results described in Chapter 3 cannot be used in the case of quasi-Monte Carlo.

Nevertheless, we elucidated an alternative way of Monte Carlo partial result validation using a trusted grid node in Chapter 3. Fortunately, this method 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, $[f_t - k\sigma_t, f_t + k\sigma_t]$, based on its mean f_t and standard deviation σ_t using similar analysis as described in Section 3.3.1. 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 6.2 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. Recomputations of such subtasks are recommended. The only problem is that with quasi-Monte Carlo, the confidence level is not necessarily known.

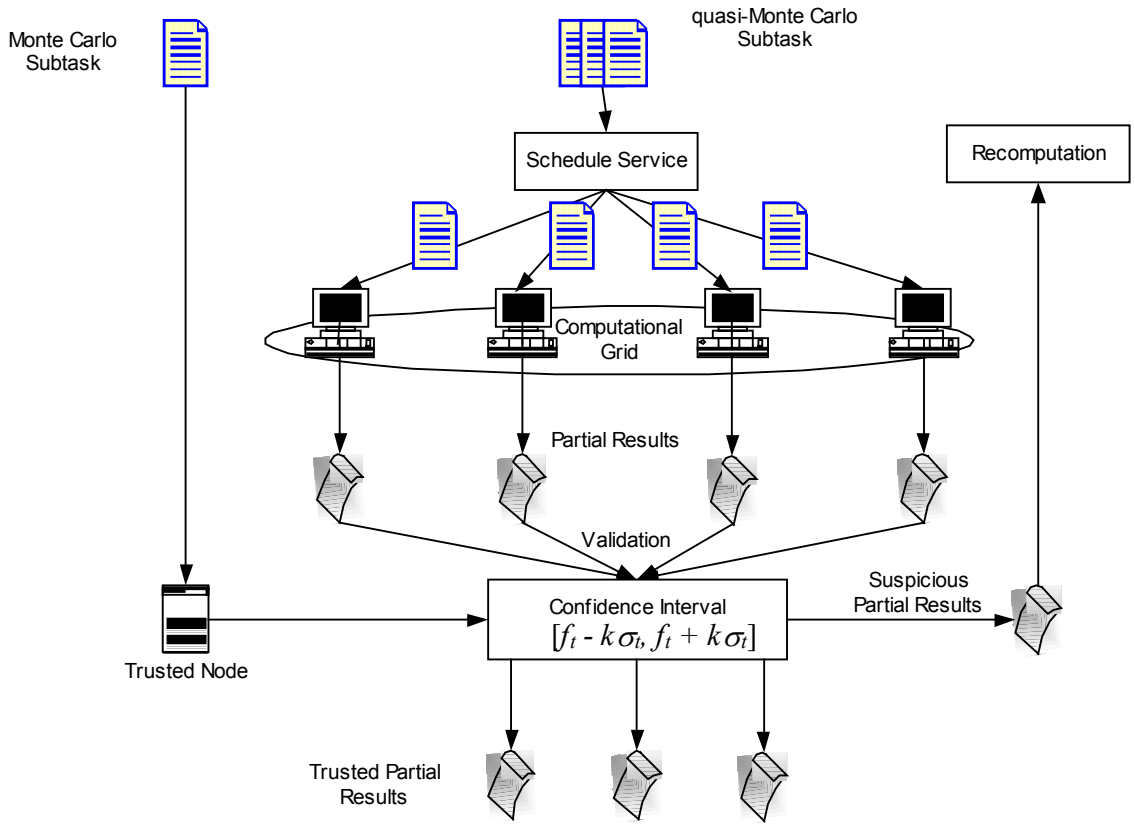

Figure 6.2: Extended Partial Result Validation Method for Grid-based Quasi-Monte Carlo applications

6.1.3 Lightweight Checkpointing and Intermediate Value Checking for Grid-based Quasi-Monte Carlo Applications

Compared to the *N-out-of-M* subtask scheduling and the partial result validation, the techniques of lightweight checkpointing and intermediate value checking are essentially the same for grid-based quasi-Monte Carlo applications. In lightweight checkpointing for grid-based quasi-Monte Carlo applications, what we need to save as checkpoint data is the current status of the quasirandom number generator and other programming related information. Leapfrog in quasirandom number sequence can also be implemented. In [51], B. C. Bromley illustrates a leapfrog scheme for the Sobol' sequences, which can actually be used in our intermediate value checking technique for grid-based quasi-Monte Carlo applications.

6.2 Extension of the use of Pseudorandom Number Generators for Intermediate Value Checking

In Chapter 3, we proposed an approach of checking the pseudorandom numbers in Monte Carlo subtasks as intermediate values to validate the faithful execution in a potentially untrusted grid node. Some pseudorandom number generators have the following properties: 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. For grid-based Monte Carlo applications, the underlying pseudorandom numbers can be naturally used as intermediate values for further validation checking. However, the nice property of these pseudorandom number generators motivates us to use them for the other applications on a computational grid. In this section, we propose an approach of "artificially" embedding pseudorandom numbers into an application to generate intermediate values for such checking.

In many other grid-based applications, random numbers are not needed for their computation. However, within grid application design, we can force a subtask of such an application to generate pseudorandom numbers during its execution. For example, the subtask can be designed to generate a certain amount of pseudorandom numbers at each step. At the same time, the value of the current pseudorandom number after every N (e.g. N = 1,000,000) pseudorandom numbers is recorded just as what is done in the grid-based Monte Carlo applications. These values become the intermediate values of the subtask. By regenerating these values using the leapfrog techniques of pseudorandom number generators on a validation server, we can match them to check if this subtask is faithfully executed on the grid node. In this approach, embedding pseudorandom numbers into the grid applications does not contribute to the computational results; however, it provides a way to enforce the trustworthiness of running the application on the grid.

6.3 Application-level Lightweight Checkpointing

The application-level lightweight checkpointing can also be extended to other applications on the computational grid, especially to those applications whose operations can be divided into many subsequent steps. Saving the application-level checkpoint data after every certain number of steps can help the application to rebuild the computational task status after interruption without wasting previous computations. As we mentioned in previous chapters, application-level checkpointing has significantly better portability and efficiency than process-level checkpointing.

The MD simulation discussed in Section 5.2 is one of the examples of using application-level checkpointing. The MD simulation uses Monte Carlo to generate the initial configuration of the host liquid. After that, the molecular dynamics at each time step is deterministic. During the procedure of the MD simulation, the water atoms' locations and velocities and other related data at the current time step are saved as checkpoint data after every certain number of time steps. The checkpoint data can then be used to rebuild the task status at the checkpointing time step so as to resume further molecular dynamics simulation.

CHAPTER 7

SUMMARY AND POSTDISSERTATION RESEARCH

Monte Carlo applications generically exhibit naturally parallel and computationally intensive characteristics, which is a natural fit for the grid-computing environment. In this dissertation, by analyzing the statistical nature of Monte Carlo applications and the cryptographic aspects of the underlying pseudorandom number generators, we developed techniques of an *N-out-of-M* subtask schedule strategy, Monte Carlo-specific lightweight checkpointing, partial result validation, and intermediate value checking. These techniques were utilized to implement grid services in our grid-computing infrastructure for Monte Carlo applications, GCIMCA, with the purpose of high-performance and trustworthy grid-based Monte Carlo computations. Grid-based Monte Carlo integration and grid-based hybrid MD/BD simulation applications were computed within a grid environment including GCIMCA. Our preliminary results show the effectiveness of our techniques for improving performance and enforcing the trustworthiness of grid-based Monte Carlo computations. Finally, these techniques are extended to a broader range of grid applications.

At the same time, this dissertation raises many interesting questions and opportunities for us to continue our study in our future post-dissertation research. The following is the list of some of these possible research directions.

1. The OGSA (Open Grid Services Architecture) is becoming the standard for grid services. In the future, we plan to adopt the emerging OGSA into GCIMCA so

that we can integrate these standard grid-computing services into grid-based Monte Carlo applications.

2. We need to analyze more grid-based Monte Carlo applications, especially "real-life" applications in engineering and industry, and apply our techniques and software infrastructure to them. We believe that these grid-computing techniques can be applied to many Monte Carlo applications in computational biology, medical science, meteorology, financial mathematics, material science, nuclear science, and mechanical engineering.

3. In Chapter 5, we discussed the implementation of a grid-based MD/BD simulation of receptor-ligand interactions in structured protein systems. Since we now have a more powerful computational application to study receptor-ligand interactions compared to the previous parallel version, we also plan to study more structured protein systems in order to predict and analyze cell signaling processes and small molecule drug efficacies. More aggressively, we expect to develop a "plug-drug" system based on our MD/BD simulation with the purpose of searching for good drug candidates.

4. In our Monte Carlo-specific lightweight checkpointing technique, the checkpoint data are stored locally. However, if the grid node becomes unavailable, the checkpoint data are still not accessible, and this leads to a waste of computational effort. We plan to implement a remote-checkpointing grid service using gSOAP [92]. This remote-checkpointing service allows a subtask to save the checkpoint data to a remote checkpoint data server. The checkpoint data file is in the XML format, and hence provides good portability.

5. In our dissertation research, we concentrated on how to serve a single Monte Carlo application using the developed grid services. In many cases, we assumed

that there are many more computational service providers than available subtasks. However, if multiple naturally parallel applications are running on the grid, how should we efficiently schedule all the available computational resources to achieve the best overall throughput? The theory of games may be appliable here.

6. In our implementation of application-level checkpointing in GCIMCA, the Monte Carlo application programmers have to specify the location of the main loop and also the changed variables that need to be saved during the checkpoint operation. This leads to increasing programming complexity. However, it might be possible to automatically analyze the program to extract the above information. This is more like compiler-related research.

7. In Chapter 6, we discussed the extension of our techniques to a broader range of grid applications. The goal here is to study the extent to which these application – specific services can enhance other grid computations. We plan to expand the functionalities and services in GCIMCA to experiment with the Monte Carlo-based services on non-Monte Carlo applications.

APPENDIX A

FAST LEAP-AHEAD PROPERTY OF PSEUDORANDOM NUMBER GENERATORS

Pseudorandom number generators exhibit the fast leap-ahead property [76], which enables us to easily and economically jump ahead in the sequence. By the fast leap-ahead algorithm, we can leap from a particular point in a pseudorandom number generator's cycle to a new point j steps away in $O(\log_2 j)$ "operations." The following paragraphs describe the implementations of fast leap-ahead algorithm in some common pseudorandom number generators.

1. Linear Congruential Generators (LCG)

The LCG is the most commonly used generator for pseudorandom numbers. It was introduced by D. H. Lehmer in 1949 [60] and is based on the following recursion:

$$X_n = aX_{n-1} + b \,(\mathrm{mod}\, M)\,. \tag{26}$$

In this expression, M is the modulus, a is the multiplier, and b is the additive constant. the corresponding leapfrog generator with leaping length j can be represented as

$$X_n = (A * X_{n-1} + C) (\mathrm{mod}\quad M)\,, \tag{27}$$

by replacing the multiplier a and the additive constant c by new values A and C, where

$$\begin{aligned} A &= a^j (\mathrm{mod}\quad M), \textit{ and} \\ C &= ca^{j-1}(a-1)^{-1} (\mathrm{mod}\quad M)\,. \end{aligned} \tag{28}$$

2. Shift-Register Generators (SRG)

SRGs are based on the following recursion:

$$X_n = \sum_{i=1}^{k} a_i X_{n-i} \pmod{M}, \tag{29}$$

where the X_n's and the a_i's are either 0 or 1 and M is 2. Matrix and polynomial methods can be used to implement fast leap-ahead in SRG sequences.

In the matrix method, with an initial vector $\mathbf{X}_0 = [X_0, X_{-1}, \ldots, X_{-k+1}]^T$ of length k, one can define a vector recursion of SRG as

$$\mathbf{X}_n = \mathbf{A}\mathbf{X}_{n-1} \pmod{2}, \tag{30}$$

where the matrix **A** has the form

$$A = \begin{Bmatrix} 0 & a_1 & a_2 & \cdots & a_{i-1} & a_i & a_{i+1} & \cdots & a_{k-2} & a_{k-1} & a_k \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{Bmatrix}. \tag{31}$$

This is a $k \times k$ matrix with elements defined modulo 2. Given the initial (seed) vector $\mathbf{X}_0$, we can compute the jth vector in the sequence as

$$\mathbf{X}_j = \mathbf{A}^j\mathbf{X}_0 \pmod{2}, \tag{32}$$

This algorithm for leaping j elements in the shift-register's sequence requires the following two steps:

1) Computation of $\mathbf{A}^j \pmod 2$, which can be accomplished with $O(\log_2 j)$ matrix-matrix multiplications, and
2) The subsequent evaluation of $\mathbf{A}^j\mathbf{X}_0 \pmod 2$, which requires a single matrix-vector multiplication.

The polynomial method is another common algorithm for computing the *j*th element of a shift-register sequence based on polynomial algebra instead of matrix algebra. First, we define the characteristic polynomial, $f(x) = a_k x^k - a_{k-1} x^{k-1} - \ldots - 1$. Then, we need to build a polynomial $r(x)$ defined as

$$r(x) = x^j \pmod{f(x)} = \sum_{i=1}^{k} c_i x_i \ . \tag{33}$$

Then, the polynomial method consists of two steps:

1) Evaluate $r(x) = x^j \pmod{f(x)}$, which can be accomplished in $O(\log_2 j)$ polynomial-polynomial multiplications, and
2) Evaluate X_j using the following equation

$$X_j = \sum_{i=1}^{k} c_i X_{i-k} \ , \tag{34}$$

where c_i are the coefficients of $x^j \pmod{f(x)}$.

3. Additive Lagged-Fibonacci Generators (ALFG)

The Additive Lagged-Fibonacci Generator (ALFG) is defined by the following recursion:

$$X_n = X_{n-k} + X_{n-l} \pmod{2^m}, k < l \ . \tag{35}$$

Similar to the implementation in SRG, the matrix and polynomial method can also be used in ALFG. In the matrix method, matrix **A** becomes

$$A = \begin{Bmatrix} & & & & & & k & & & & & l \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{Bmatrix}, \tag{36}$$

and the modulo is 2^m instead of 2. In the polynomial method, $f(x)$ turns out to be $f(x) = x^l - x^{l-k} - 1$ and the modulo is 2^m.

4. Multiplicative Lagged-Fibonacci Generators (MLFG)

The recurrence relation for MLFG is defined by the following equation, which is closely related to that for the ALFG:

$$X_n = X_{n-k} \times X_{n-l} \pmod{2^m}, k < l. \tag{37}$$

Actually, we can get the following equation for the MLFG [96]:

$$X_n = (-1)^{Y_n} 3^{Z_n} \pmod{2^m}, \tag{38}$$

where Y_n is given by the recurrence,

$$Y_n = Y_{n-k} + Y_{n-l} \pmod{2}, \tag{39}$$

and Z_n by the recurrence,

$$Z_n = Z_{n-k} + Z_{n-l} \pmod{2^{m-2}}. \tag{40}$$

The fast leap-ahead implementation in Makino's scheme [93] consists of, equivalently, choosing seed Y_0, …, Y_{l-1}, and maximal period seed Z_0, …, Z_{l-1}, and then computing Y_j and Z_j efficiently in $O(l^2 \log j)$ time using Miller and Brown's algorithm. Once Y_j and Z_j are computed, X_j can be easily obtained by equation (38).

5. Inversive Congruential Generators (ICG)

The ICGs have two versions, the recursive ICG is given by the formula,

$$x_{n+1} = a\bar{x}_n + b \pmod{m}, \tag{41}$$

while the explicit ICG is defined as follows:

$$x_n = \overline{an+b} \pmod{m}. \tag{42}$$

Here $\bar{c}$ denotes the multiplicative inverse of a modulo m in the sense that $c\bar{c} \equiv 1 \pmod{m}$ when $c \neq 0$, and $\bar{0} = 0$.

The fast leap-ahead in recursive ICG can be implemented using the following method. We define a sequence $c_0, c_1, \ldots,$ where

$$\begin{aligned} c_0 &= 0 \\ c_1 &= 1 \\ c_{n+2} &= bc_{n+1} + ac_n \pmod{m} \end{aligned} \tag{43}$$

By induction [94], we also obtain that

$$x_n = \bar{c}_n c_{n+1} \pmod{m}. \tag{44}$$

Then, the x_{n+j} can be generated by

$$x_{n+j} = \bar{c}_{n+j} c_{n+j+1} \pmod{m}, \tag{45}$$

where c_{n+j+1} and c_{n+j} can be obtained by

$$\begin{pmatrix} c_{n+j+1} \\ c_{n+j} \end{pmatrix} = \begin{pmatrix} b & a \\ 1 & 0 \end{pmatrix}^j \begin{pmatrix} c_{n+1} \\ c_n \end{pmatrix}. \tag{46}$$

In (46), we can pick up $c_n = 1$ and $c_{n+1} = x_n$ for computational convenience. This requires only $O(\log_2 j)$ multiplications of 2×2 matrices to compute c_{n+j+1} and c_{n+j}.

The implementation of fast leap-ahead in EICG is relatively easy. We can simply substitute n in (42) with $n + j$ [65].

6. Combined Generators

Combining different recurrences can increase the period length and improve the structural properties of pseudorandom generators. By combining the output of the basic generators, a new random sequence can be constructed of the form:

$$z_n = x_n \circ y_n,$$

where ° is typically either the exclusive-or operator or addition modulo some integer m or addition of floating-point random numbers modulo 1, and x and y are random number sequences from two different types of random number generators. If both x_n and y_n have the fast leap-ahead property, then z_n can also leap ahead with $O(\log_2 j)$ operations.

REFERENCES

1. H. Casanova, "Distributed Computing Research Issues for Grid Computing," ACM SIGACT, **33**(2), 2002.

2. D. Anderson, D. Werthimer, J. Cobb, E. Korpela, M. Lebofsky, SETI@home: Internet Distributed Computing for SETI, Bioastronomy 99: A New Era in Bioastronomy, G. Lemarchand and K. Meech, eds, ASP Conference Series No. 213 (Astronomical Society of the Pacific: San Francisco), pp. 511, 2000.

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

4. "Distributed.net," http://www.distributed.net, 2002.

5. Y. Li, M. Mascagni, "Grid-based Monte Carlo Applications," Lecture Notes in Computer Science, **2536**:13-24, GRID2002, Grid Computing Third International Workshop/Conference, Baltimore, 2002.

6. J. J. Duderstadt and L. J. Hamilton, "Transport Theory," Wiley & Sons, 1976.

7. J. Spanier and E. M. Gelbard, "Monte Carlo Principles and Neutron Transport Problems," Addison-Wesley, Reading, Massachusetts, 1969.

8. A. R. Leach, "Molecular Modeling: Principles and Applications," Pearson Education Ltd., 2001.

9. G. S. Fishman, "Monte Carlo: Concepts, Algorithms, and Applications," Springer-Verlag New York, 1995.

10. C. E. Leith, "Theoretical skill of Monte Carlo Forecasts,' Monthly Weather Report, **102**:409-418, 1974.

11. J. M. Hammersley and D. C. Handscomb, "Monte Carlo Methods," Spottiswoode, Ballantyne & Co Ltd, 1964.

12. J. Basney, R. Raman and M. Livny, “High Throughput Monte Carlo,” in Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, Texas, March 22-24, 1999.

13. Y. Li, M. Mascagni, “Analysis of Large-scale Grid-based Monte Carlo Applications,” invited paper in a special issue of the International Journal of High Performance Computing Applications (IJHPCA), 2003.

14. D. Tal, N. Rishe, S. Navathe, and S. Graham, “On Parallel Architecture,” the proceedings of PARBASE-90, 1990.

15. A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, P. F. Reynolds, “Legion: The Next Logical Step Toward a Nationwide Virtual Computer,” Technical report No. CS-94-21, June, 1994.

16. Z. Nemeth and V. Sunderam, “A Formal Framework for Defining Grid Systems,” proceedings of the second IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin, 2002.

17. D. B. Skillicorn, “Motivating Computational Grids,” proceedings of the second IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin, 2002.

18. United Device, “The History of Distributed Computing,” http://www.ud.com/products/dc/history.htm.

19. M. Litzkow, M. Livny, and M. Mutka, “Condor - A Hunter of Idle Workstations,” Proceedings of the 8th International Conference of Distributed Computing Systems, pages 104-111, June, 1988.

20. M. Livny, J. Basney, R. Raman and T. Tannenbaum, “Mechanisms for High Throughput Computing,” SPEEDUP Journal, 11(1), 1997.

21. J. Basney and M. Livny, “Deploying a High Throughput Computing Cluster”, High Performance Cluster Computing, Rajkumar Buyya, Editor, Vol. 1, Chapter 5, Prentice Hall PTR, 1999.

22. A. K. Lenstra and M. S. Manasse, “Factoring with two large primes,” Advances in Cryptology -- Eurocrypt '90, Lecture Notes in Computer Science, **473**:72-82, Springer-Verlag, Berlin, 1991.

23. GIMPS website, http://www.mersenne.org/prime.htm, 2002.

24. PiHex website, http://www.cecm.sfu.ca/projects/pihex.

25. "Folding@home Distributed Computing," http://folding.stanford.edu, 2000.

26. "Clickworkers Study," http://clickworkers.arc.nasa.gov, 2002.

27. "Project: Cancer Research," http://members.ud.com/projects/cancer, 2002.

28. SETI@home, "SETI@home: the Search for Extraterrestrial Intelligence," http://setiathome.ssl.berkeley.edu, 2002.

29. I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, 1999.

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

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

32. 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.

33. M. O. Neary, B. O. Christiansen, P. Cappello and K. E. Schauser, "Javelin: Parallel Computing on the Internet." Future Generation Computer Systems, Elsevier Science, Amsterdam, Netherlands, **15**: 659-674, 1999.

34. Platform Computing, "LSF: Load Sharing Facility," http://www.platform.com/products/wm/LSF/index.asp, 2002.

35. A. S. Grimshaw and W. A. Wulf, "The Legion vision of a worldwide virtual computer," CACM, **40**(1):39-45, 1997.

36. I. Foster, C. Kesselman, "Globus: A metacomputing infrastructure toolkit," International Journal of Supercomputer Applications, **11**(2), 1997.

37. Entropia, "PC Grid Computing," http://www.entropia.com, 2002.

38. "Jini Network Technology: An Executive Overview," Sun Microsystems Inc., 1999.

39. Globus website, http://www.globus.org, 2002.

40. Gusto Website, "http://www.globus.org/research/testbeds.html," 2002.

41. Centurion Website, “http://www.cs.virginia.edu/~legion/centurion/Centurion.html ”, 2002.

42. NASA Information Power Grid Website, “http://www.nas.nasa.gov/About/IPG/ipg.html,” 2002.

43. ASCI DISCOM Website, “http://www.llnl.gov/asci/discom,” 2002.

44. Grid Forum Website, “http://www.gridforum.org,” 2002.

45. I. Foster, C. Kesselman, J. M. Nick, Steven Tuecke, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” draft document at http://www.globus.org/research/papers/ogsa.pdf, 2002.

46. B. Dupire, “Monte Carlo Methodologies and Applications for Pricing and Risk Management,” Risk Publications, 1999.

47. S. M. Ross, “A First Course in Probability,” 2nd Ed. Macmillan, New York, 1996.

48. M. P. Allen, D. J. Tildesley, “Computer Simulation of Liquids,” Chapter 4, Oxford University Press, New York, 1987.

49. M. H. Kalos and P. A. Whitlock, “Monte Carlo Methods,” a Wiley-Interscience Publication, 1986.

50. S. Ulam, “On the Monte Carlo Method,” Proc. 2nd Symp. Large-scale Digital Calculating Machinery, 207-212, 1951.

51. B. C. Bromley, “Quasirandom Number Generators for Parallel Monte Carlo Algorithms,” Journal of Parallel and Distributed Computing, **38**:101-104, 1996.

52. R. E. Calfisch, “Monte Carlo and Quasi-Monte Carlo Methods,” A. Numerica, pp. 1-49, 1998.

53. B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, “CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations,” J. Comp. Chem. **4**:187-217, 1983.

54. J. A. Rathkopf "PMC: A shared short-cut to portable parallel power," Lawrence Livermore National Laboratory, Livermore, CA, UCRL-112311, 1992.

55. MESYST website, http://www.nea.fr/abs/html/nea-1534.html, 2002.

56. Z. Yang and B. Rannala, "Bayesian Phylogenetic Inference using DNA Sequences: a Markov Chain Monte Carlo Metho," Molecular Biology and Evolution, **14**:717-724, 1997.

57. Y. Li and M. Mascagni, "A Web-based Distributed Monte Carlo Integration Tool," Proceedings of the First Southern Symposium on Computing, 1998.

58. M. H. Zhou, "A Scientific Computing Tool for Parallel Monte Carlo in a Distributed Environment," Ph.D.'s dissertation in Scientific Computing Program, School of Mathematical Sciences, University of Southern Mississippi, 2000.

59. F. Solms and W. H. Steeb, "Distributed Monte Carlo integration using CORBA and Java," International Journal of Modern Physics, **9(7)**:903-915, 1998.

60. P. L'Ecuyer, "Uniform Random Number Generations: A Review," Proceedings of the 1997 Winter Simulation Conference, IEEE Press, 127-134, Dec. 1997.

61. M. Mascagni, "Serial and Parallel Random Number Generation," Quantum Monte Carlo in Physics and Chemistry, Peter Nightingale and Cyrus Umrigar, editors, NATO ASI Series, Series C: Mathematical and Physical Sciences, Vol. X, Kluwer, Dordrecht, pp. 277-288, 1999.

62. M. Mascagni and A. Srinivasan, "Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation," ACM Transactions on Mathematical Software, **26**:436-461, 2000.

63. SPRNG website, http://sprng.cs.fsu.edu, 2002.

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

65. A. Srinivasan, D. M. Ceperley, and M. Mascagni, "Random Number Generators for Parallel Applications," to appear in Monte Carlo Methods in Chemical Physics, D. Ferguson, J. I. Siepmann and D. G. Truhlar, editors, Advances in Chemical Physics series, Wiley, New York, 1997.

66. R. Buyya, S. Chapin, and D. DiNucci, "Architectural Models for Resource Management in the Grid," the First IEEE/ACM International Workshop on Grid Computing (GRID 2000), Springer Verlag LNCS Series, Germany, Bangalore, India, 2000.

67. Y. Li, M. Mascagni, "Improving Performance via Computational Replication on a Large-Scale Computational Grid," accepted and to appear in the proceedings of the

GP2PC at the IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE/ACM CCGRID2003, Tokyo, 2003.

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

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

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

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

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

73. XML website, http://www.xml.org.

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

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

76. M. Mascagni, "Polynomial versus Matrix Methods for Leap-Ahead in Shift-Register Type Pseudorandom Number Generators," preprint, 1998.

77. P. L'Ecuyer and S. Cote, "Implementing a Random Number Package with Splitting Facilities," ACM Transactions on Mathematical Software, **17**:98, 1991.

78. S. Aluru, G. M. Prabhu, and J. Gustafson, "A Random Number Generator for Parallel Computers," Parallel Computing, **18**, 1992.

79. Y. Li, M. Mascagni, R. van Engelen "GCIMCA: A Globus and SPRNG Implementation of a Grid Computing Infrastructure for Monte Carlo Applications",

submitted to the International Multiconference in Computer Science and Computer Engineering, 2003.

80. S. Brunett, K. Czajkowski, I. Foster, S. Fitzgerald, A. Johnson, C. Kesselman, J. Leigh, and S. Tuecke, "Application Experiences with the Globus Toolkit," Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing, 1998.

81. Y. Li, M. Mascagni, M. H. Peters, "Grid-based Nonequilibrium Multiple-Time Scale Molecular Dynamics/Brownian Dynamics Simulations of Ligand-Receptor Interactions in Structured Protein Systems," accepted and to appear in the proceedings of the First BioGrid Workshop at the 3rd IEEE/ACM Symposium Cluster Computing and the Grid, Tokyo, 2003.

82. Y. Zhang, Y. Li, M. H. Peters, "Nonequilibrium, Multiple-Time Scale Simulations of Ligand-Receptor Interactions in Structured Protein Systems," accepted and to appear in Proteins: Structure, Function, and Genetics, 2003.

83. M. H. Peters, "Nonequilibrium molecular dynamics simulation of free-molecule gas flows in complex geometries with application to Brownian motion of aggregate aerosols," Physical Review E, **50**(6):4609-4617, 1994.

84. M. H. Peters, "Fokker-Planck equation and the grand molecular friction tensor for coupled rotational and translational motions of structured Brownian particles near structured surfaces," Journal of Chemical Physics, **110**(1):528-538, 1999.

85. M. H. Peters, "Fokker-Planck Equation, Molecular Friction, and Molecular Dynamics for Brownian Particle Transport near External Solid Surfaces," Journal of Statistical Physics, **94**:557-586, 1999.

86. M. H. Peters, "The Smoluchowski Diffusion Equation for Structured Macromolecules near structured surfaces," Journal of Chemical Physics, **112**:5488-5498, 2000.

87. Y. Zhang, "Implementation of a Hybrid Molecular-Brownian Dynamics Simulation on a D-Peptide/HIV Protein Macromolecular System," Master's Thesis, Florida State University, 2001.

88. Protein Data Bank website, http://www.rcsb.org/pdb, 2002.

89. SOLVATE website, http://www.mpibpc.gwdg.de.

90. K. Toukan, A. Rahman, "Molecular-dynamics study of Atomic Motions in Water," Physical Review, **31**:2643-2648, 1985.

91. P. L'Ecuyer and C. Lemieux, “Recent Advances in Randomized Quasi-Monte Carlo Methods,” in Modeling Uncertainty: An Examination of Stochastic Theory, Methods, and Applications, M. Dror, P. L'Ecuyer, and F. Szidarovszki, editors, Kluwer Academic Publishers, pp. 419-474, 2002.

92. R. A. van Engelen and K. A. Gallivan, “The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks,” proceeding of ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid'02), Berlin, Germany, May, 2002.

93. J. Makino, “Lagged-Fibonacci Random Number Generator on Parallel Computers,” Parallel Computing, **20**:1357-1367, 1994.

94. H. Niederreiter, “Random Number Generation and Quasi-Monte Carlo Methods,” CBMS 63, SIAM, 1992.

BIOGRAPHICAL SKETCH

The Author was born in Guangzhou, China on Sept. 22, 1974. He got his B. S. in Computer Science and Engineering at the South China University of Technology in 1997. From 1997 to 1998, he worked for IBM China Ltd. as a Software and Networking IT Specialist. Then, he decided to continue his graduate study and spent a year in the program of Scientific Computing at the University of Southern Mississippi. He transferred to the Florida State University and obtained his M.S. in Computer Science in 2000. He is interested in the research of Grid computing, parallel and distributed computing, Monte Carlo methods, random number and quasirandom number generation, and computational biology.