High Throughput Monte Carlo

Jim Basney[*] Rajesh Raman[*] Miron Livny[*]

Abstract

We present a cost-effective framework for conducting large scale Monte Carlo (MC) studies which exploits the natural parallelism of the MC method to harness the power of large, dynamic collections of computing resources. We describe the benefits of the dynamic master-worker (MW) parallel programming paradigm and how task parallel and job parallel MW applications fit into our framework. We discuss the issues involved in supporting MC applications in this framework, including random number generation, resource management, remote file access, and checkpointing. We conclude with descriptions of selected customer experiences.

1 Introduction

Monte Carlo (MC) applications are naturally parallel, since random samples may be computed and analyzed independently. The Condor High Throughput Monte Carlo (HTMC) framework delivers the computing power of CPUs distributed across a network to scientists conducting large scale MC studies. This framework combines the dynamic master-worker (MW) paradigm with a distributed, scalable random number generator and the services provided by a High Throughput Computing (HTC) environment [1].

The paper proceeds as follows. Section 2 defines the dynamic MW paradigm, and section 3 describes the random number generator used to implement our framework. In section 4, we describe in detail the services needed by HTMC applications and how those services are provided by the Condor HTC environment. We conclude with a description of user experiences in section 5.

2 Dynamic Master-Worker Paradigm

The master-worker paradigm is a natural fit for parallel Monte Carlo applications, since random samples may be computed and analyzed independently. In a MW application, the master partitions the goal, assigns a sub-goal to each worker, and receives results when workers complete their assigned task. The master may use application-specific knowledge when determining goal partitioning and sub-goal dissemination policies. MW applications may be task parallel or job parallel.

We define a task parallel application to be a multi-threaded or multi-process application where one thread or process acts as the master, which is responsible for directing and coordinating the computation. The master partitions the goal computation and sends sub-goals to worker entities, which perform the required processing and send the results back to the master via shared-memory or some other inter-process communication mechanism (network sockets, pipes, etc.). Since the processing capacity available to the application may change and the complexity of the computation over the sample space may vary, it

[*]Computer Sciences Department, University of Wisconsin-Madison

is often desirable to allow the application to quantitatively and qualitatively adapt the worker pool. Additionally, since the goals assigned to workers are typically independent, the master may be implemented to tolerate worker failures. When the failure of a worker is detected, the master can reassign that worker's computation to another.

We define a job parallel application to be an application which obtains parallelism by submitting many jobs to a batch scheduler. The master creates input files for each worker and submits each worker as a job. The master detects job completion and collects the output files. The master may submit many more workers than there are nodes available in the batch system, relying on the scheduler to manage the queued jobs. If a worker fails, the master may re-submit that job to the batch scheduler.

While a HTC environment strives to hide the dynamic nature of the environment from its customers as much as possible, better throughput levels may be obtained by agile parallel applications that themselves embrace and adapt to changes in the environment. In particular, the parallel application should adapt to changes in the number of available resources, the types of available resources, and resource preemptions or failures. For this reason, we have found the dynamic MW architecture to be a natural fit in the HTMC framework.

3 Random Number Generation

While Monte Carlo applications are widely regarded as "embarrassingly parallel," the degree of application parallelism that affects the quality of results is heavily predicated on the quality of the random number streams employed by the application. Parallel MC applications rely heavily on the availability of statistically independent streams of random numbers to significantly decrease the variance of the calculation.

To facilitate a higher degree of parallelism for MC applications, we are working to integrate the SPRNG [2] scalable pseudorandom number generation library into our framework. The SPRNG library provides a portable interface to several high quality random number generators and strives to minimize correlation between streams. To allow reproducibility and convenient assignment of random number streams, a global name space for streams is defined. The master assigns streams by name to specific workers, simplifying the management and dissemination of unique random streams.

4 High Throughput Computing Services

The accuracy of the results of MC studies and the confidence associated with these results increases with the amount of computation invested in exploring the problem space. Thus, scientists and engineers who conduct MC studies have a seemingly infinite appetite for computing power and are continuously on the lookout for additional computing resources. A HTC environment harnesses the computing capacity available in pools of networked resources and conveniently delivers a sustained high level of computing throughput to its customers.

Large collections of resources experience continuous evolution as system software and hardware are upgraded or replaced. System failure may also occur in such pools, which requires some resources to be temporarily withdrawn from the pool while they are repaired. In addition, older components may be permanently removed from the resource pool. To provide high throughput over long periods of time, a HTC environment must adapt and continue to correctly operate in such dynamic environments.

For over a decade, the Condor Team at the University of Wisconsin-Madison has been developing and deploying mechanisms and frameworks for HTC [3]. Our work has been guided by close interactions with groups of HTC users that include scientists and engineers from a wide range of disciplines. These users have been using services provided by Condor to simulate a wide spectrum of phenomena including diesel engines, neural networks, high energy physics events, computer hardware and software, the behavior of crystals, and randomized optimization techniques.

The Condor system is in production use at many sites around the world. At the University of Wisconsin-Madison, Condor currently harnesses the power of more than 600 workstations scattered throughout campus to support the computational needs of real-life MC applications. The most important lesson our experience has taught us is that in order to deliver and sustain high throughput over long time intervals, the framework must build its resource management services on an integrated collection of robust, scalable and portable mechanisms. Robustness minimizes down time whereas scalability and portability increases the size of the resource pool a study can draw upon. Typical HTC environments are physically distributed and distributively owned, meaning that the control over powerful computing resources is distributed among many individuals and small groups. The owners use the owned resource for their daily needs and define the resource's usage policy for other customers. Resources in HTC pools may include dedicated workstations as well as commodity personal computers, leading to large heterogeneous environments. The natural evolution of the resource pool coupled with the physical and policy heterogeneity lead to a dynamic environment that requires robust resource management mechanisms. Fragile mechanisms that depend on the unique characteristics of specific computing platforms are likely to have a negative rather than a positive impact on the long term throughput of a MC study.

4.1 Resource Management

An expressive resource management architecture enables the HTMC framework to adapt to changes in the resource pool. In a dynamic, task parallel master-worker application, the master may continuously interact with the HTC environment resource management interface to look for new resources and to learn about the resources currently available. For example, the master may know that certain phases of the MC computation require a workstation with a large amount of physical memory while other phases do not. The master can use this information to request both workstations with large physical memories and those with small physical memories. When resources are allocated, the master can match each workstation with a worker which will use it effectively. Similarly, in a job parallel MW application, the master may specialize each job definition, so some jobs require workstations with a large amount of physical memory while other jobs don't.

In Condor, each customer is represented by a customer agent which manages a dynamic queue of job descriptions and sends resource requests to the matchmaker [4]. One job may request many resources, as in the case of a task parallel MW application. Each resource is represented by a resource agent, which implements the policies of the resource owner and sends resource offers to the matchmaker. The matchmaker is responsible for finding matches between resource requests and resource offers, and notifying the relevant agents when a match is found. Upon notification, the customer agent and the resource agent perform a claiming protocol to initiate the allocation. This architecture is illustrated in figure 1.

Resource requests and offers contain a description of the request (or offer), a constraint

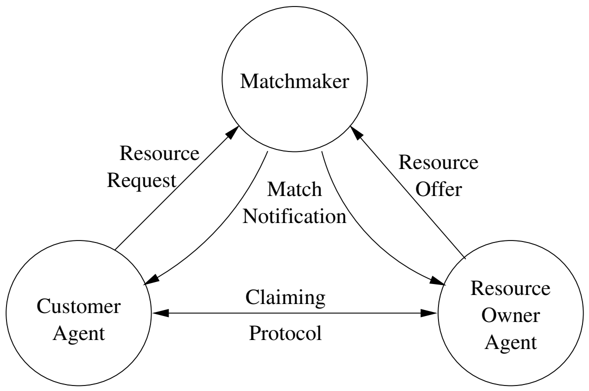

FIG. 1. *Condor Resource Management Architecture*

which specifies which matches are acceptable, and a rank which acts as a "goodness metric" that defines preferences among compatible matches. For example, the customer agent may advertise a resource request which describes itself as a request owned by a particular user and requires only resources running a specific operating system, with a preference for resources with larger memories. Similarly, the resource agent may also include a constraint in the resource offer which specifies if it is available to service a request, and which requests it will service. For example, in accordance to the resource usage policy defined by the resource owner, the resource agent may only be willing to service requests made by specific customers, with a preference for applications with small memory image sizes. An offer matches a request when both constraints are satisfied.

The matchmaker implements system-wide policies by imposing its own set of constraints and preferences on matches. For example, the matchmaker implements a customer priority mechanism by matching resource requests in priority order, so resource requests from customers with better priorities have a better opportunity to find a match. Although the allocation is established between the resource and customer agents through the claiming protocol, the matchmaker may preempt allocations by matching a resource with a new request to maintain a fair distribution of allocations. The customer agent or the resource agent may also choose to break the allocation at any time.

This architecture separates the advertising, matchmaking, and claiming protocols. The agents advertise resource offers and requests asynchronously to the matchmaker, and the matchmaker notifies the agents when a match is found. Since the matchmaker is not involved in the claiming protocol, the claiming protocol may be customized for specific types of agents, and may be modified without effecting the matchmaker, which remains a neutral and generic service. The matchmaker does not need to know the details of allocation establishment, and so many different allocation protocols may be easily supported by the same matchmaker [4].

Two methods are provided for the master of a MW application to interact with the Condor scheduler. In the case of a task parallel application, the master contacts the customer agent to modify the job description to add, remove, or alter requests for resources. The customer agent then notifies the master when resources are allocated or deallocated. This may be accomplished using the standard PVM interface or via Condor extensions to the PVM interface for additional expressibility [5]. Thus, many MW applications written in PVM naturally interface with the Condor scheduler without modification. In the case of a job parallel application, the master may use standard Condor system commands to submit and remove jobs. The Condor scheduler writes a log file of job events, and an API allows the master to easily process this log for event notification. For example, this method may be used by the master to determine when a job has completed.

4.2 Resource Discovery

Qualitative resource discovery by the application increases the ability of the master of a master-worker application to make smart placement decisions when dispatching work to worker nodes. For example, if the master discovers that a fast machine with a large memory has become available, it may decide to migrate a long running worker to the newly available node. Thus, an agile MW application that can discover resources can continuously be on the lookout for better resources, and thus be supplied with resource information that guides its goal partitioning and sub-goal dissemination policies.

The matchmaking framework described thus far is also used as the foundation for resource discovery. The resource offers sent by resource agents include a detailed representation of the characteristics of the resource such as its physical memory, virtual memory, performance ratings (KFLOPS and MIPS), architecture, operating system, current load average, keyboard idle time, etc. Our framework allows applications to query the matchmaker for available resources, and like resource requests, the query may be expressed with constraints and ranks defined over arbitrary attributes of resource offers.

4.3 File Management

A HTMC application must have efficient access to its data files. For job parallel master-worker applications, the file system is the primary communications mechanism between master and worker. The amount and timing of file I/O varies significantly between MC applications, which suggests that one file access mechanism will not be the best fit for all MC applications. We consider four mechanisms for remote data access: I/O system call redirection, network file systems, application-level network data access, and data file staging.

To redirect file I/O system calls, we use an interposition agent that injects itself between the application and the operating system and services file I/O system calls itself [6, 7], as illustrated in figure 2. This is accomplished by linking the application with an interposition library. The Condor environment invokes a remote procedure call to perform the file operation on a server with access to the customer's data files. Since the file operations are performed at the system call level, this may result in many high latency operations, reducing the performance of the application. Read-ahead and write-behind caching can effectively reduce this latency. Redirecting file I/O system calls has the significant benefit that it places no file system requirement on the remote workstation. This enables the Condor environment to utilize a greater number of resources. Additionally, the remote file access is transparent to the application. Only a re-link with a special library is required.

If all nodes in the cluster share a network file system, Condor may be configured to allow applications to use the network file system for remote data access. One benefit of using a network file system is that the application need not be re-linked with a system call interposition library. This allows Condor to support commercial MC applications and others which can not be re-linked. File access is also transparent in this scenario. The availability, reliability, and performance of the network file system depends on the type of network file system used (NFS, AFS, etc.) and how it is deployed and supported in the organization. In our experience, network file systems are often optimized for small reads

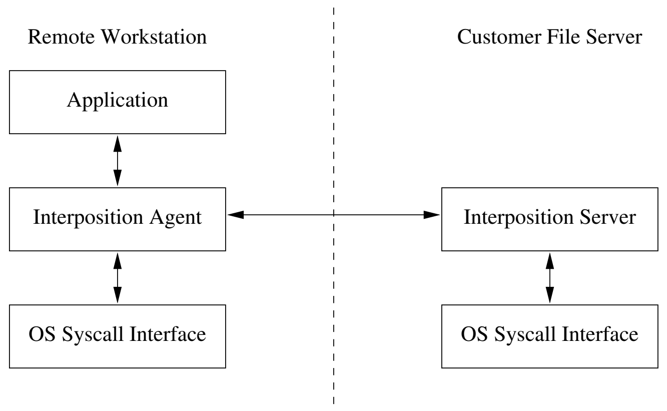

Fig. 2. *System Call Interposition*

and writes, so performance can be poor for I/O intensive MC applications. Also, since using a network file system restricts the application to only those workstations which have access to the network file system, this solution may significantly restrict the computing resources available for the MC computation.

Remote data access may alternatively be implemented at application-level using network communication libraries (PVM, MPI, Berkeley sockets, etc.) or database access libraries. This solution can often be the most efficient, since the data access protocol may be customized for the application, and it can also be portable to many platforms when using a portable network communication or database access library. However, this method of remote data access can cause problems for checkpointing, as described below in section 4.4.

To implement data file staging, the Condor environment requires a list of input files from the customer for each application. The system then transfers these input files to the local disk of the remote workstation before running the application. The system is responsible for gathering up the application's output files and transferring them to a destination specified by the customer when the application has completed. This requires free disk space on the remote workstation and the bandwidth to transfer the data files at the start and end of each allocation. For large data files, this results in high start-up and tear-down costs compared to a block file access mechanism provided by a network file system or redirected file I/O system calls.

Condor scheduling services may also be used to improve data access locality. For example, data sets may be distributed among nodes in the Condor cluster. Each resource may advertise the data sets available locally, and each job may require (or prefer) a resource with the appropriate data sets available locally. This is particularly useful for MC applications with very large input data files.

4.4 Checkpointing

A long-running dynamic master-worker application must be prepared for worker (and possibly master) failures. The lost work which results from these failures may significantly reduce the throughput obtained by the application. To ameliorate this problem, the application may utilize a mechanism by which partially completed work can be preserved. Checkpointing provides such a service.

A checkpoint of an executing program is a snapshot of its state which can be used to restart the program from that state at a later time. Checkpointing provides reliability:

when a compute node fails, the program running on that node can be restarted from its most recent checkpoint, either on that same node once it is restored or potentially on another available node. At the same time, checkpointing also enables preemptive-resume scheduling. All parties involved in an allocation can break the allocation at any time without losing the work already accomplished by simply checkpointing the application. Thus, a long running application can make progress even when allocations last for relatively short periods of time. Condor provides a fully integrated and robust checkpointing mechanism [8] in the form of a library that can be linked with the application. Checkpoint servers provide the storage space needed to store checkpoint files, and the communication overhead of moving these files to and from the server is considered by the Condor scheduler.

Checkpointing processes which use network communication requires that the state of the network be checkpointed and restored. Our checkpointing library has been enhanced to support applications which use PVM or MPI [9]. To checkpoint the state of the network, this library synchronizes communicating processes by flushing all communication channels prior to checkpoint. At restart time, the library restores the communication channels. However, processes which access the network using other interfaces (for example, Berkeley sockets) are currently not fully supported by our checkpointing services.

The transparent checkpointing services provided by Condor are effective for most applications. However, some system calls are not currently supported by Condor's checkpointing services, so applications which use these system calls must find an alternative. Additionally, checkpointing large applications can require significant network and disk resources, which may be unacceptable in some environments. The Condor scheduler does not require applications to use the Condor checkpointing services. One alternative is for an application to provide its own application-level checkpointing. The Condor scheduler can be configured to trigger application-level checkpointing at preemption time by sending a specified Unix signal. Application-level checkpointing is often more efficient than transparent checkpointing, since the application may choose to save only the minimal state required to resume computation later. A second alternative is to distribute short pieces of work in the MW application. In this case, when a worker fails or is preempted, only a short amount of work is lost. The work may be assigned to a new worker in a task parallel application, and in a job parallel application, the preempted job will be rescheduled to begin the work again when an available node is located by the Condor scheduler.

5 Experience

Our research has benefitted greatly from the feedback provided by customers who use the Condor environment for production Monte Carlo runs. This feedback has helped us fine-tune our framework to better meet the needs of real MC applications. We present here a sample of our customers' experiences.

A group of researchers from the University of Amsterdam has been running its MC application in six Condor pools located in three different countries and spanning two continents. Over the last three years they have used more than 160 CPU years to search for global potential energy minima of a N-particle system consisting of Lennard-Jones particles on a spherical surface. At any given time, hundreds of workers of the above application could have been found scattered over the different pools. Some of these workers consumed more than 100 days of CPU over a lifetime of 4–5 months. In many cases these workers were left unattended as members of the group were away from their desks attending meetings or on vacation. The group expected the HTC environment not to lose any of these workers before or during their execution phase. Like most other HTC users we have worked with, they counted on the robustness of the mechanisms used by the environment to successfully take their workers from submission to completion.

A group of physicists at the INFN (Italian National Institute for Nuclear Physics) has been using Condor for MC experiments since 1996. They ran a simulation program based on GEANT 3.21 in Condor pools in Italy and the United States as part of the WA92 event simulation experiment at CERN (European Laboratory for Particle Physics). It involved the complete event simulation and reconstruction under various experimental situations in several channels of physics. The INFN is now constructing a wide-area network Condor pool for HTMC. This pool will consist of more than one hundred workstations distributed throughout Italy, including compute sites in Bologna, Rome, Milan, and Naples.

Researchers at the Massachusetts Institute of Technology use Condor to run a simulation which studies how the dynamic interactions of multiple planets can be important in the evolution of other planetary systems. They typically run several hundred MC integrations of possible planetary systems just after they have crossed the stability boundary, all in parallel on Condor. They then evolve each system until the system reaches a stable configuration, often requiring millions of (simulated) years. The simulation performs job parallel MC integrations by assigning slightly different initial conditions to each job. This experiment thus far has consumed more than 2,500 CPU days in six months. They plan to perform as many as 10,000 more numerical integrations over the next year on the Condor system.

Scientists at NCSA (National Computational Science Alliance) have been using Condor to perform large-scale Quantum Monte Carlo (QMC) calculations of electronic structure of molecular systems. QMC is a method for solving the Schrodinger equation using stochastic approaches, so that the complicated many-body effects are be described directly and accurately. The important advantage of QMC is that the calculations can be run very efficiently in parallel as the communication can be almost completely avoided by appropriate algorithm modifications. Their calculations have consumed more than 1,300 CPU days in nine months.

A researcher at the University of Wisconsin Engine Research Center has been using Condor to model two-phase, turbulent, reacting flows within the combustion chambers of diesel engines with KIVA [10]. He performs the simulations in multiple phases. First, a course grained simulation is run to find points of interest. Then, the points of interest are investigated using a more precise simulation. In a two week period, this researcher consumed approximately 30 CPU days each day using Condor.

A professor of physics at the University of Wisconsin has been using Condor to study magnetic order on a 2D "kagome" lattice with MC simulations of different lattice sizes to extrapolate to the macroscopic limit. This study has a large parameter space and many MC steps per site are required for statistical significance. The runs are broken into segments for different machines. The lower limit required for equilibrium behavior is approximately 40 independent runs of $5\text{x}10^5$ steps per site (10^7 total steps) and lattices ranging from 16x16 to 64x64 for each parameter set and eight temperatures near Tc. Due to the volume of data, the submission command file is generated automatically, as well as the data file averaging and error analysis. Command line arguments are used in the submission file to eliminate large numbers of input files.

6 Conclusion

We have presented the Condor High Throughput Monte Carlo framework and described how this framework may be effectively used for MC studies. To provide sustained high throughput over long periods of time, our framework effectively adapts to changes in large, dynamic collections of computing resources. Many of our customers have found this framework to be very successful at meeting the needs of their MC applications.

7 Acknowledgements

We are grateful to the scientists mentioned in this paper for providing descriptions of their Condor experiences and to the developers of SPRNG for sharing their parallel pseudorandom number generation expertise with us.

References

[1] M. Livny and R. Raman. High-throughput resource management. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 13. Morgan Kaufmann Publishers, Inc., 1998.

[2] M. Mascagni, D. Ceperley, and A. Srinivasan. Sprng: A scalable library for pseudorandom number generation. In *Proceedings of the Third International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, June 1998.

[3] M. J. Litzkow and M. Livny. Experience with the condor distributed batch system. *IEEE Workshop on Experimental Distributed Systems*, 1990.

[4] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 1998.

[5] J. Pruyne and M. Livny. Interfacing condor and pvm to harness the cycles of workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[6] M. J. Litzkow. Remote unix, turning idle workstations into cycle servers. In *Proc. of the 1987 Usenix Summer Conf.*, pages 381–384, 1987.

[7] M. Jones. Interposition agents: Transparently interposing user code at the system interface. *14th ACM Symposium on Operating Principles*, 27(1), December 1993.

[8] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and migration of unix processes in the condor distributed processing system. University of Wisconsin-Madison Computer Sciences Technical Report 1346, 1997.

[9] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *Workshop on Job Scheduling Strategies for Parallel Processing*, 1996.

[10] A. Amsden, J. Ramshaw, P. O'Rourke, and J. Dukowicz. Kiva: A computer program for two- and three-dimentional fluid flows with chemical reactions and fuel sprays. Los Alamos National Laboratory Report LA-10245-MS, February 1985.