Simulation of Emission Tomography Using Grid Middleware for Distributed Computing

M. G. Thomason, R. F. Longton, J. Gregor
Department of Computer Science
University of Tennessee
Knoxville, TN 37996 USA
{thomason,longton,jgregor}@cs.utk.edu

G. T. Smith, R. K. Hutson
Department of Radiology
University of Tennessee Medical Center
Knoxville, TN USA 37920
{gsmith,khutson}@mc.utmck.edu

November 11, 2002

TR-CS-02-491
Department of Computer Science
University of Tennessee
Knoxville, TN USA 37996

Abstract

SimSET is Monte Carlo simulation software for emission tomography. This paper describes a simple but effective scheme for parallel execution of SimSET using NetSolve, a client-server system for distributed computation. NetSolve (version 1.4.1) is "grid middleware" which enables a user (the client) to run specific computations remotely and simultaneously on a grid of networked computers (the servers). Since the servers do not have to be identical machines, computation may take place in a hetergeneous environment. To take advantage of diversity in machines and their workloads, a client-side scheduler was implemented for the Monte Carlo simulation. The scheduler partitions the total decay events by taking into account the inherent compute-speeds and recent average workloads, i.e., the scheduler assigns more decay events to processors expected to give faster service and fewer decay events to those expected to give slower service. When compute-speeds and sustained workloads are taken into account, the speed-up is essentially linear in the number of equivalent "maximum-service" processors. One modification in the SimSET code (version 2.6.2.3) was made to ensure that the total number of decay events specified by the user is maintained in the distributed simulation. No other modifications in the standard SimSET code were made. Each processor runs complete SimSET code for its assignment of decay events, independently of others running simultaneously. Empirical results are reported for simulation of a clinical-quality lung perfusion study.

Keywords: distributed computing, emission tomography, grid middleware, Monte Carlo simulation, parallel computing

1 Introduction

Monte Carlo simulation of aspects of emission tomography is well-established (cf [1, 2, 3, 4, 5]). SimSET [6], a simulation system for emission tomography, is a public domain package of routines written in C. SimSET is summarized at its web site [7] as follows:

> The SimSET package uses Monte Carlo techniques to simulate the physical processes and instrumentation in emission imaging...
>
> The Photon History Generator ... performs Monte Carlo simulations of photon creation and transport through heterogeneous attenuators for both SPECT and PET, i.e., it generates photons and transports them through the "object" to the face of the collimator or detector.
>
> The Collimator Module receives... and tracks photons through the collimator being modelled.
>
> The Detector Module... tracks photons through the specified detector, recording the interactions within the detector for each photon. The interactions are used to compute a detected location and total energy deposited.
>
> The Binning Module is used to process photon and detection records... Histograms and images, together with photon history files, make up the final output from a simulation....

A realistic simulation may require millions of emission decay events. This may result in an extremely long execution time when a simulation is run as conventional sequential computation on a single processor. But the individual decay events are stochastically independent; therefore, if multiple processors are available, the heavy computational load can be partitioned into subcomputations that run in parallel on different processors [8, 9], the outputs of which are combined upon completion.

One way to access multiple processors is to use a parallel computer [10, 11]. Another way is to implement distributed computation on a network of computers—a mode of computation in which the individual machines do not have to be identical [12]. We implemented distributed computation of SimSET by using NetSolve [13, 14], a public domain package of routines in C (plus some environment-specific interfaces) which enables a user to run application-specific code remotely on networked computers. An introduction to NetSolve at its web site [15] states:

> ...NetSolve... provides remote access to computational resources, both hardware and software.
>
> The NetSolve system is comprised of a set of loosely connected machines. By *loosely* connected, we mean that these machines are on the same local, wide, or global area network, and may be administered by different institutions and organizations. Moreover, the NetSolve system is able to support these interactions in a *heterogeneous* environment, i.e., machines of different architectures, operating systems, and internal data representations can participate in the system at the same time...
>
> NetSolve and systems like it are often referred to as Grid Middleware...
>
> No root/supervisor privileges are needed to install or use any component of the NetSolve system....

In the context of this paper, the phrase "grid middleware" refers to the fact that a user who wants a particular computation to be executed remotely goes through NetSolve for service on a grid of networked machines.

This paper is organized as follows. Section 2 describes NetSolve with emphasis on characteristics relevant to our implementation of distributed computation of SimSET. Section 3.1 describes the Unix process for SimSET that runs on servers, and section 3.2 describes the way in which simultaneous runs of SimSET are automatically scheduled on different servers. Section 4 reports experimental results for simulation of 100 million decay events in a clinical-quality lung perfusion study.

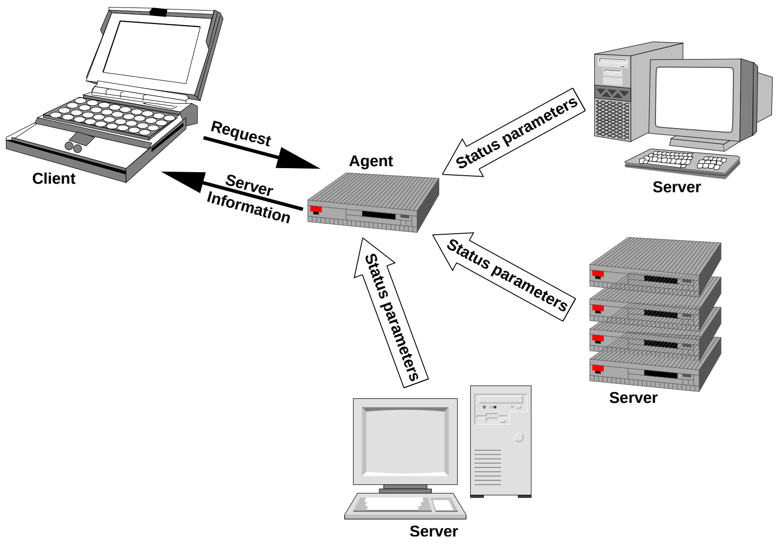


Figure 1: NetSolve client contact with agent for multiple, heterogeneous servers

2 NetSolve Organization

NetSolve [13, 14, 15] is organized as a client-server system in which a *client* obtains information from an *agent* before sending requests-for-computation to *servers*. The NetSolve client library is linked with the user's application code; thereafter, the client-user can transmit requests for specific computations to occur remotely on servers. The client's request goes first to a NetSolve agent—an information service that maintains a database about the capabilities and status of a set of (possibly heterogeneous) servers. Upon receipt of a request from its client, the agent responds with data about servers and indicates its choice of a server for the job.

Figure 1 represents the client's contact with its agent, after which the client sends its request along to a server. The compute-engines in a NetSolve system are its servers. These are the networked computers running the server daemon that enables them to receive and process requests from clients and to send status reports to agents. Figure 1 shows a heterogeneous system in which two servers are conventional workstations but the third is a machine consisting of multiple, identical processors.

Each server has code for a set of specific computations. These computations are the NetSolve *problems* that the server can run locally on request. The client's request must specify one of these problems and supply the inputs that an instance of the problem needs. Each problem is documented in a NetSolve *problem description file* (a PDF) which contains such information as the name by which the problem is known in the NetSolve system, the lists of I/O-parameters passed between client and server, and the calling sequence that a client must use. To expand the suite of problems already computable on various servers, a user or system administrator must create a PDF for the new problem and install the code on one or more servers. We created a PDF to run SimSET on Unix[1] servers as the process described in section 3.1.

NetSolve uses TCP/IP and a customized application layer protocol for communication among its components. If more than one NetSolve server is available for the client's problem and the client makes a server-nonspecific request, the request goes to the server selected by the agent; however, the client can designate a particular server by using the server-host machine's IP address.

An instance of a NetSolve problem can be requested in *blocking mode* (the application sending the request is idle while waiting for the server to finish and return values) or in *non-blocking mode* (the application

[1]Unix means Unix-like, including Solaris and Linux.

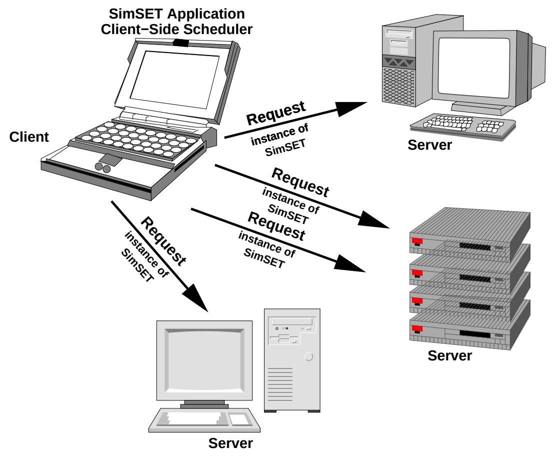


Figure 2: NetSolve client requesting multiple instances of SimSET on diverse servers

can continue with other computations and periodically test a flag that indicates whether the server has finished). In order to have computations running in parallel, the client sends a series of non-blocking requests, one after another, for instances to run simultaneously on NetSolve servers. Figure 2 represents the client requesting four instances of SimSET to run simultaneously. Two servers are single-processor workstations and the third is a multiprocessor machine. The requests are for simultaneous runs on each workstation's processor and two processors in the third server.

An aspect of distributed computation is that processors on different servers may provide computational service to the client at different relative rates. From the perspective of the client-user, this service is a function of the time to move data to/from the server, the time taken by the processor for actual computation on its instance of the problem, and some additional NetSolve overhead. Neither the additional overhead (measured consistently as less than a few hundred milliseconds in similar applications [14]) nor the data-transfer times have a major impact on service in our SimSET implementation. However, the relative service of a processor when running SimSET depends both on the raw compute-speed of the server-host machine (a constant) and on its current workload (a variable). In networks of computers, it is common to have some processors that are inherently slower than others and some with significantly larger workloads than others at the time the application needs them. As an integral part of distributed computation of SimSET, we implemented a *client-side scheduler* which can partition a large simulation into multiple instances, each instance being an independent run of SimSET on a different processor for a fraction of the total number of decay events. The fractions do not have to be equal because the scheduler can assign larger fractions to processors with higher expected service rates.

In this sense, the distributed computation of the SimSET application is statically *self-scheduled* using server-status data obtained from the NetSolve agent each time the application is run. The scheduler partitions the total decay events so that each instance of the problem is expected to take about the same wall-clock time, given the information about servers available to the scheduler at the time it determines the partition. The client-side scheduler is discussed further in section 3.2.

We installed SimSET version 2.6.2.3 (with one modification described in section 4) as a NetSolve problem on a system running NetSolve version 1.4.1. The application creates multiple, non-blocking requests for the simultaneous runs scheduled automatically by the client-side scheduler. Technical details and empirical results for this system are given in section 4.

3 SimSET as a NetSolve Problem

3.1 Process on Unix Servers

SimSET for computation on Unix servers is a NetSolve problem with a user-defined PDF. The complete SimSET code runs locally on a processor as a Unix process. All I/O is data from/to the application (the simulation I/O is not germane to NetSolve itself). I/O is redirected standard input, output, and error, plus any additional files opened by the program.

A processor on a Unix server with SimSET code executes as follows for an instance of the problem:

(1) A Unix process is started to read in a file sent by the client-user with the information needed by the server for instances of SimSET. This file includes the parameters for the SimSET run and the complete paths for relevant files.

(2) Using the fork() system command, the process forks into a parent process and a child process.

- The child process opens and redirects standard input, output, and error, and opens any other files. It then runs the instance of SimSET according to the information read in.
- The parent process uses the wait() system command to wait for the child process to signal the end of SimSET execution. Upon completion, the Unix process for the instance of SimSET terminates.

3.2 Client-Side Scheduling

In distributed computation in heterogeneous environments, “machines of different architectures, operating systems, and internal data representations can participate in the system at the same time....” [15]. A consequence is that some machines may be inherently faster than others on identical code, and the random differences in run-time workloads may further spread the relative service rates at the time the client needs resources. *Client-side scheduling* can react to this diversity by imposing an uneven partition of decay events.

The client-side scheduler for SimSET is C code included in the user’s application. Its function is to partition a simulation into smaller instances of the problem and distribute those instances to different processors for simultaneous execution. Each time the application is run, the client-side scheduler:

(1) contacts the agent and obtains, for each potential server, the latest information about status and workload that the agent provides;

(2) partitions the simulation into smaller instances, not necessarily equal in the number of decay events, for assignment to the processors to have the same expected wall-clock time of completion for all instances.

The scheduler determines the partition by estimating the service each processor will provide, relative to the maximum service, in the following way. Let N be the number of processors available for an application that requires D decay events in total. Suppose first that all N processors have no other jobs (have a workload of 0) when the user wants to run SimSET. Let C_i denote the speed-factor of processor i relative to the fastest processor when running one SimSET process and no additional jobs; i.e., C_i for $0 < C_i \leq 1$ is a relative measure of the service of processor i when unloaded—a benchmark constant for a machine with a workload of 0. Thus, a processor j with $C_j = 1$ and workload 0 is a “maximum-service” processor. A slower processor k for which $C_k = 0.8$ (for example) can provide service equivalent to 80% of the maximum at best. If all N processors always had workload 0, the scheduler could simply compute $S = C_1 + C_2 + \cdots + C_N$ and partition the simulation into N instances such that the instance for processor i is simulation of $D_i = DC_i/S$ decay events.

But the scheduler also takes recent information about server workloads into account. The NetSolve agent provides to the client the recent system load average for each server. Let L_i denote this average for server i. In NetSolve, a new value of L_i is sent by server i to notify the agent of a noteworthy increase or decrease in its workload [14]; specifically, server i sends a new L_i whenever the percentage change in value exceeds a fixed reference. The agent retains the most recent value of L_i and provides that value to the client. L_i for the multi-processor servers discussed in section 4 is the machine-average workload and applies to each processor in the server.

In an environment of varying workloads, L_i is the latest information available to the scheduler at the time it must create the partition. Assuming that the average L_i is representative of near-term loading, the fractional usage of processor i which the scheduler expects for its SimSET process is $f_i = 1/(1 + L_i)$, $0 < f_i \leq 1$. To account for this expected fraction as well as the speed-factor C_i, the scheduler computes $S = C_1 f_1 + \cdots + C_N f_N$ and partitions the simulation into N instances of SimSET such that processor i is instructed to simulate $D_i = DC_i f_i/S$ decay events.

Various options (such as the number of decay events to simulate), paths, and file names for a simulation are input to the SimSET code in the Parameter File [7]. Having determined the partition, the scheduler creates a Parameter File for each instance and fills in these Files appropriately for servers selected. The client-user then sends the requests to the servers, one request immediately after another. The scheduler does not dynamically adjust the partition or migrate work from one server to another while the instances are running. The total time taken by the scheduler after receiving data from the NetSolve agent is negligible in comparison with other computations.

4 Empirical Results

The computational resources used for the results in this paper consist of three NetSolve servers: two Sun V880 symmetrical multiprocessor machines and a Sun Blade 2000 workstation. All three use the Solaris operating system. Both V880s are equipped with eight 750MHz UltraSPARC III CPUs and 32 Gbytes of primary memory. The Blade 2000 has a 900MHz UltraSPACR III CPU and 2 Gbytes of primary memory. This means that up to 17 processors can be used for SimSET. The servers and the Unix client machine are interconnected via Gigabit Ethernet.

Our method of distributed computation requires one change to the SimSET code. The change is made to ensure that partitioning of the SimSET computation as described above will not result in simulating fewer decay events in total than the user specifies in an application. The number of decay events to be simulated by an instance of SimSET is one of the simulation options in its Parameter File. Let D denote that number, and let V denote the number of source activity voxels. To apportion an integer-number of decay events among an integer-number of voxels, the original code truncates D to the nearest value $\widehat{D}$ for $\widehat{D} \leq D$ such that $\widehat{D} = kV$ where k is an integer. We replaced the truncation operation with a randomized rounding scheme specifically to ensure that each source activity voxel is assigned k decay events corresponding to $\widehat{D}/V$. In addition, we use the fraction $D/V - k$ to randomly decide on a voxel-by-voxel basis whether to assign an extra decay event to a voxel. This approach agrees with the method SimSET already uses when the number of decay events D is less than the number of source activity voxels V. The important implication of the change is that partitioning the SimSET computation still results in an integer-count of decay events very close the number specified in the application. The random number generation implemented in SimSET [7] was not changed, namely, a first-level random number generator creates an array of seeds for a second-level generator. The client-side scheduler ensures a different initial seed for the first-level generator for each

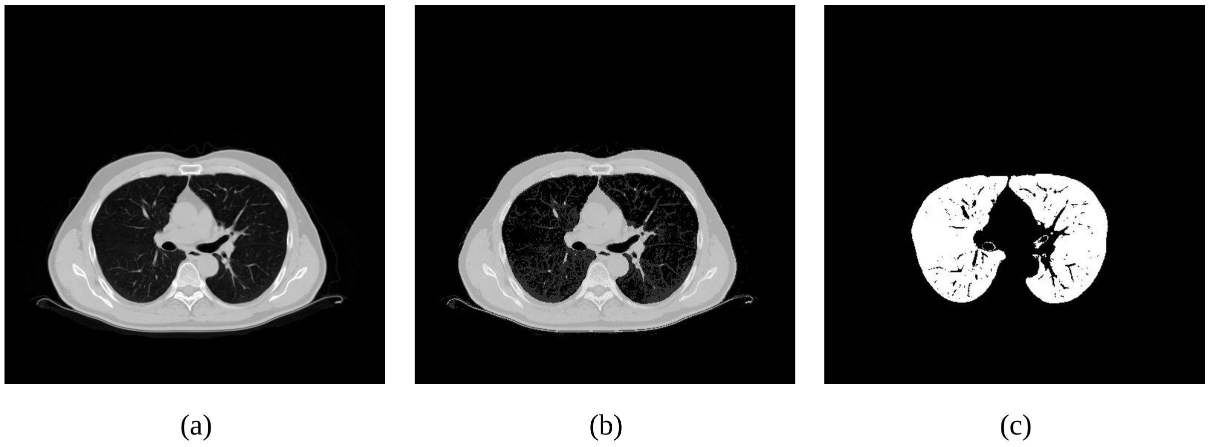


Figure 3: One slice of the input image data used for the experiments: (a) original x-ray CT image of a human chest; (b) and (c) the corresponding SimSET attenuation and source activity maps.

instance of the SimSET problem. (Parallel random number generation itself is an active topic of research [16].)

The two main data inputs to SimSET are a set of attenuation maps and a set of source activity maps. We use clinical quality x-ray computed tomography (CT) chest images for the former and simulate the latter as described below. The x-ray CT data set consists of 116 image slices, each 512x512 with an axial (between-slices) resolution of 3mm and a transaxial (within-slice) resolution of approximately 1mm. To obtain SimSET-compatible attenuation maps, each gray level value is mapped into the closest corresponding type of tissue supplied with SimSET. See Figure 3(a) and (b) for illustrations.

The source activity maps represent a clinical nuclear medicine lung perfusion study in which a patient is given a dose of Tc-99m macroaggregated albumin (MAA) intraveneously. In a normal patient, the MAA will be distributed relatively uniformly through out the lung fields. To generate corresponding source images, we segment each x-ray CT image to obtain a binary image that represents uniform MAA activity in the lung parenchyma. All x-ray CT images are segmented using the same threshold parameters. See Figure 3(c) for an illustration.

Planar lung perfusion studies are based on obtaining eight views of the patient using a gamma camera with an acquisition window of 140 ± 14 keV. Typically, five to six million photons are detected for the entire study which translates into approximately 100 million decay events. Figure 4 shows the eight planar projection images produced by one run of SimSET for the attenuation and source activity maps described above. Each image is 128x128 with a spatial resolution of approximately 3mm. The images are similar in quality to those in clinical nuclear medicine practice.

We conducted two experiments to measure time-of-computation on the networked servers. In experiment I, we ran the SimSET lung perfusion simulation for 100 million decay events, first using a single V880 processor, then using two, three, ..., up to all eight on one server. The V880 was not running any other jobs (its non-SimSET workload L_i was 0). We measured start-up times associated with NetSolve and found them to be neglible. More importantly, we measured the runtimes associated with SimSET. Figure 5 plots these numbers normalized relative to the time of a single processor. The plot shows speed-up very close to linear. A caveat, of course, is that if too few decay events are being simulated, the SimSET start-up overhead will constitute a relatively large portion of the overall runtime—and notably less than linear speed-up will result.

In experiment II, we used all 17 processors to simulate the $D = 100$M decay events. We first determined the benchmark speed-factor C for each kind of processor by initial runs of 10 million decay events using one (unloaded) processor on each machine. These empirical numbers are within 1% of the corresponding

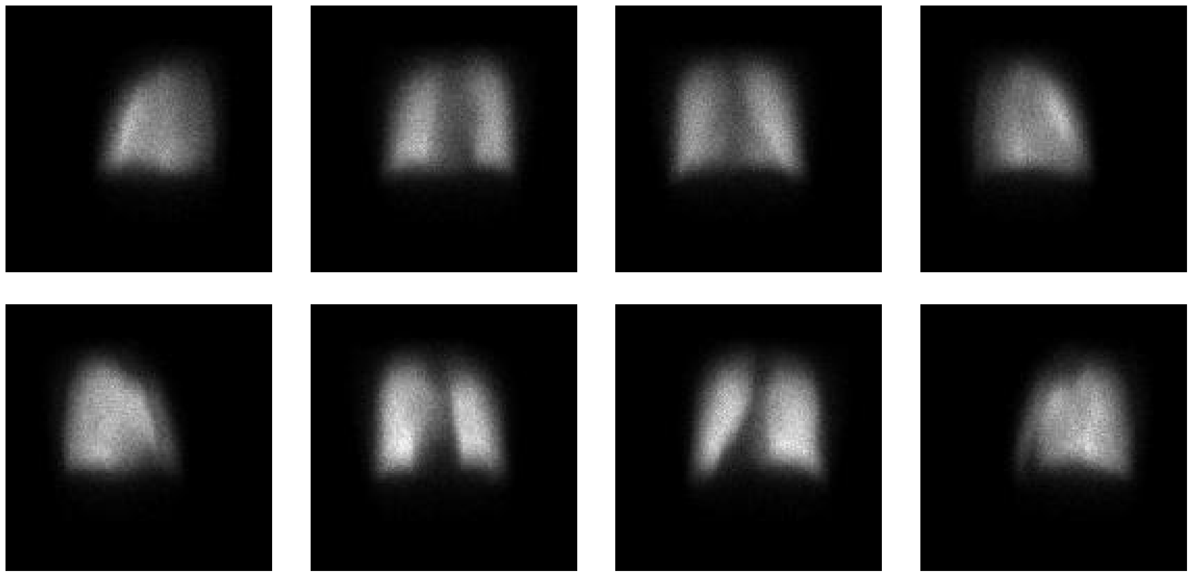


Figure 4: Simulated planar projection images produced by SimSET.

CPU clock-rate ratios, i.e., $C = 1$ for the 900MHz CPU and $C = 0.833$ for the 750MHz CPUs. Thus, the reference for maximum service is the workstation processor running one SimSET process and no other jobs. One V880 processor can provide service equivalent to 83.3% of that maximum if it also has workload 0.

To demonstrate the impact of diverse workloads on scheduling, we then created sustained (non-SimSET) workloads of $L_i = 1.5$ for one V880, $L_j = 0.5$ for the second V880, and $L_k = 1$ for the workstation. The corresponding fractions used by the scheduler are $f_i = 0.4$, $f_j = 0.67$, and $f_k = 0.5$. Multiplying these fractions by the benchmarks $C_i = C_j = 0.833$ and $C_k = 1$ respectively, the scheduler obtains

$C_i f_i = 0.333$ for each processor on one V880,
$C_j f_j = 0.556$ for each processor on the other V880, and
$C_k f_k = 0.5$ for the workstation processor.

This is equivalent to approximately 7.6 maximum-service processors for the simulation.

Figure 6(a) shows the scheduler's assignment of decay events to the 17 processors. Processors 1-8 are the more heavily loaded V880 ($C_i f_i = 0.333$), processors 9-16 are the less heavily loaded V880 ($C_j f_j = 0.556$), and processor 17 is the Blade 2000 workstation ($C_k f_k = 0.5$). Processor 17 is assigned proportionally fewer decay events to simulate than processors on the V880 with smaller average workload, but proportionally more decay events than processors on the V880 with larger average workload. Figure 6(b) shows the overall wall-clock times for all 17 processors. The minimum and maximum are 2,523 and 2,560 seconds respectively, with the mean and standard deviation being 2,545 and 12.5 seconds. That the minimum, maximum, and mean are so close to one another indicates successful client-side scheduling under the conditions described.

To place the wall-clock times for the distributed SimSET code in context, simulating 100 million decay events on one unloaded V880 processor in experiment I took 6 hours and 8 minutes. Under the sustained workloads in experiment II, the 17 processors are equivalent to about 7.6 maximum-service processors, and the distributed computation on them averages about 43 minutes. If all 17 machines are unloaded, we have the equivalent of 14.33 maximum-service processors, in which case the simulation of 100 million decay events takes approximately 22 minutes.

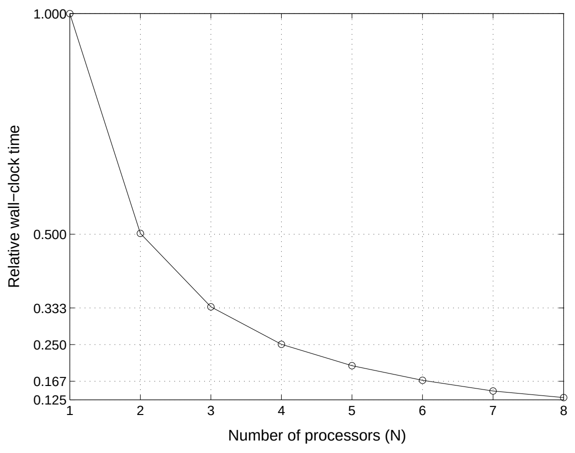
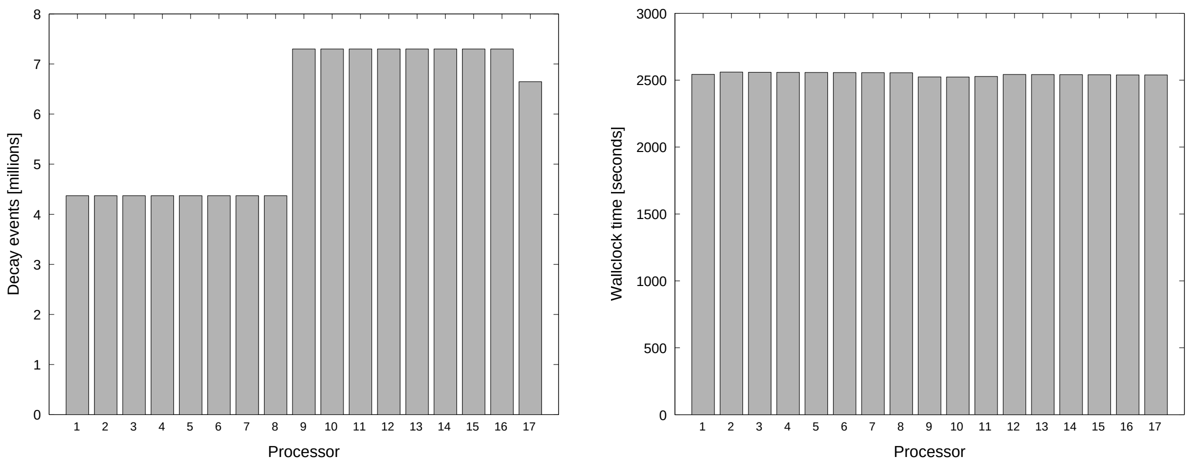


Figure 5: Relative SimSET wall-clock time as a function of number of unloaded V880 processors used.



(a) (b)

Figure 6: (a) The client-side scheduler uses inherent speed-factors and load averages to partition the total number of decay events into unequal assignments, with (b) the end result being execution times that are close to identical.

5 Summary

The thrust of this paper is not validation of SimSET per se; rather, the focus is on distributing the computation across a grid of heterogeneous machines to achieve significantly faster computation. By making the complete simulation a NetSolve problem, the user avoids having to make configuration-specific changes within the SimSET code.

We implemented SimSET version 2.6.2.3 in a distributed-computing environment using NetSolve version 1.4.1. One modification was made in the way SimSET apportions decay events to voxels to preserve the total number of decay events specified by the user.

In general, it is not the case that all servers in a network provide the same relative rate of computational service on a problem. The service in the work reported here is determined essentially by the inherent compute-speed of a processor and its current workload (other factors being negligible in comparison). A client-side scheduler was implemented to make the SimSET application statically self-scheduling, in the sense that the scheduler partitions the total number of decay events and assigns a portion to each available processor. The fraction assigned to an individual processor is proportional to the processor's benchmark speed-factor and its recent average workload. When the speed-factors and sustained workloads are taken into account, the speed-up achieved is essentially linear in equivalent maximum-service processors.

An objective of grid middleware projects like NetSolve is to provide substantial computing resources to a user's application on a temporary, as-needed basis with transparency and ease-of-access. Rapid developments not only in hardware, but also in organizational concepts and distributed-resource management, make distributed computation increasingly viable and attractive for biomedical computing.

Acknowledgements

Dr. Robert Harrison of the University of Washington provided insightful comments with respect to the change made in the SimSET code. The SInRG project [17] (National Science Foundation Grant EIA 9972889) at the University of Tennessee–Knoxville provides infrastructure and resources for research in grid-based computing.

References

[1] M. Ljungberg and S-E Strand, "A Monte Carlo Program for the Simulation of Scintillation Camera Characteristics," *Comp. Mtds. and Progs. in Biomed.*, vol. 29, pp. 257-272, 1989.

[2] C. J. Thompson, C. J. Moreno, and Y. Picard, "PETSIM: Monte Carlo Simulation of All Sensitivity and Resolution Parameters of Cylindrical Positron Imaging Systems," *Phys. Med. Biol.*, vol. 37, pp. 731-749, 1992.

[3] M. F. Smith, C. E. Floyd, and R. J. Jaszczak, "A Vectorized Monte Carlo Code for Modeling Photon Transport in SPECT," *Med. Phys.*, vol. 20, pp. 1121-1127, 1993.

[4] M. Ljungberg, S-E Strand, and M. A. King, Eds, *Monte Carlo Calculations in Nuclear Medicine: Applications in Diagnostic Imaging*, Institute of Physics Pub., Bristol, UK, 1998.

[5] H. Zaidi, A. H. Scheurer, and C. Morel, "An Object-Oriented Monte Carlo Simulator for 3D Cylindrical Positron Tomographs," *Comp. Mtds. and Progs. in Biomed.*, vol. 58, pp. 133-145, 1999.

[6] T. K. Lewellen, R. L. Harrison, and S. Vannoy, "The SimSET Program," Chapt. 7, pp. 77-92, in [4].

[7] SimSET Web Site **http://depts.washington.edu/∼simset/html/simset_home.html**, July 2002.

[8] W. R. Martin, “Monte Carlo Methods on Advanced Computer Architecture,” *Adv. Nucl. Sci. Tech.*, vol. 22, pp. 105-164, 1992.

[9] W. R. Martin, T. C. Wan, T. Abdel-Rahman, and T. N. Mudge, “Monte Carlo Photon Transport on Shared Memory and Distributed Memory Parallel Processors,” *Int. Jour. Supercomp. Appl.*, vol. 1, pp. 57-74, 1987.

[10] H. Zaidi, C. Labbe, and C. Morel, “Implementation of an Environment for Monte Carlo Simulation of Fully 3-D Positron Tomography on a High-Performance Parallel Platform,” *Parallel Comp.*, vol. 24, pp. 1523-1536, 1998.

[11] Y. K. Dewaraja, M. Ljungberg, A. Majumdar, A. Bose, and K. F. Koral, “A Parallel Monte Carlo Code for Planar and SPECT Imaging: Implementation, Verification, and Applications in ^{131}I SPECT,” *Comp. Mtds. and Progs. in Biomed.*, vol. 67, pp. 115-124, 2002.

[12] D. R. Kirkby and D. T. Delpy, “Parallel Operation of Monte Carlo Simulations on a Diverse Network of Computers,” *Phys. Med. Biol.*, vol. 42, pp. 1203-1208, 1997.

[13] H. Casanova and J. Dongarra, “NetSolve: A Network-Enabled Server for Solving Computational Science Problems,” *Int. Jour. Supercomp. Appls. and High Perform. Comp.*, vol. 11, pp. 212-223, 1997.

[14] D. C. Arnold, H. Casanova, and J. Dongarra, “Innovations of the NetSolve Grid Computing System,” to appear in *Concurrency—Practice and Experience.*

[15] NetSolve Web Site **http://icl.cs.utk.edu/netsolve**, July 2002.

[16] M. Mascagni and A. Srinivasan, “SPRNG: A Scalable Library for Pseudorandom Number Generation,” *ACM Trans. Math. Software*, vol. 26, pp. 436-461, 2000.

[17] UTK Scalable Intracampus Research Grid Web Site **http://icl.cs.utk.edu/sinrg**, July 2002.