REMOTE++: A Script for Automatic Remote Distribution of Programs on Windows Computers

Ashley Hopkins
Department of Computer Science and Engineering
4202 East Fowler Avenue, ENB 118
University of South Florida
Tampa, FL 33620
amhopki2@csee.usf.edu

Abstract

Execution of simulation programs requires large amounts of CPU resources. Parallel Independent Replications (PIR) is one method of reducing simulation run time by enabling time-based parallelization of simulations on multiple distributed machines. Existing PIR systems are targeted for Unix machines. A Windows-based program for distributing executables and their associated input and output files to idle Windows PCs is developed. Wintel PCs dominate desktop computing, therefore, providing hours of unused CPU cycles that can be exploited to accelerate simulation. The new REMOTE++ program uses standard remote shell (rsh) and remote copy (rcp) services to enable remote program execution and the transfer of input and output files. The remote computer needs only a standard rsh /rcp daemon available from independent software vendors. A single master computer maintains a job list and host list and distributes jobs (from the job list) to remote hosts (from the host list). Execution results are on the master computer at the completion of a remote execution. Status from remote executions is maintained in a log file on the master computer. The REMOTE++ program builds on a previous REMOTE version and improves reliability and performance. The REMOTE++ program was used to investigate run time trends needed for steady state simulation of an M/M/1 queue at utilizations approaching 1.0. It was found that as the utilization approaches 100% the simulation time grows increasingly longer. The REMOTE++ program is open source and freely available from the authors.

Categories and Subject Descriptors

I.6.m [Simulation and Modeling]

General Terms

Performance, Experimentation

Keywords

Remote distribution, parallel independent replications, simulation

1. INTRODUCTION

Remote distribution and parallelization of programs can be used to reduce execution time. One method for speeding execution is time parallelization. Time parallelization is used to speed the execution of programs that require numerous runs to complete a single experiment. For example the simulation of a queue requires multiple runs with different input parameters and control variables to determine its behavior. Time parallelization would enable this program to be executed on several computers at the same time, each with different input values to reduce the overall run time of the simulation. Parallel Independent Replications (PIR) uses time parallelization of programs. PIR is used to distribute programs to multiple machines to be run in parallel. Typically PIR is used to distribute multiple instances of the same program using different input parameters as described above. PIR can also be used to distribute different executables to run in parallel. This allows multiple parameters to be held constant. Through parallel execution of these programs, a greater number of input and control variables can be evaluated, which provides more output and ultimately more accurate results.

There exist few PIR tools for Windows PCs. In this paper we develop and evaluate such a tool called REMOTE++ (building on [2]). The remainder of this paper is organized as follows. Section 2 describes existing Unix-based tools for PIR. Section 3 overviews the goals of a PIR tool. Section 4 describes the design and implementation of REMOTE++. Section 5 contains an evaluation of REMOTE++. Section 6 is a summary and describes future work.

2. REVIEW OF EXISTING METHODS

The parallel execution of programs on remote computers to speed execution of a simulation is not a new idea [5] [7]. Batch systems have long been used to distribute processes to remote machines. The majority of existing methods for remote execution are implemented using Unix. Since the majority of computer users have Windows PCs, unused computer resources are predominately available on these machines. Unix has standard rsh (remote shell) and rexec (remote execution) commands, which enable easy distribution of processes. Windows supports these rsh and rexec commands, but only to distribute programs to Unix machines. Windows does not include the daemon that must be present on each remote computer for the commands to execute. To enable the use of these standard commands, independent vendors have daemons available for Windows machines.

2.1 Review of Remote Execution in Unix

Condor [5] uses the idle CPU cycles of a network of workstations. Condor was initially developed for Unix. In Unix, Condor operates on the processors of idle machines then migrates the job to another workstation when the user of a remote computer returns. Checkpoints are used to migrate the job and start execution at close to the same point where it was interrupted on the prior machine. Condor has been available on UNIX platforms since 1988 and became available for Windows NT/2000/XP in the version 6.4.3 release in October of 2002. Condor for Windows [3] does not have the migration capabilities it has under Unix. It can suspend execution when the user of a remote machine returns, but must start execution from the beginning on the next machine.

Akaroa [6] is a project developed by the University of Canterbury to use multiple parallel processors to speed up quantitative stochastic simulations. The Akaroa project takes a different approach; it runs ordinary serial simulations on multiple parallel processors and continuously collects the observations. These observations are then averaged and, when sufficient data is collected, the simulation can be halted. Akaroa is portable to most variants of Unix, but cannot be used on a Windows platform.

2.2 Review of GRID Computing

Another method of sharing resources is Grid computing [1] [4]. Grid computing is an approach to distributed systems that shares resources over a local or wide area network. Grid computing attempts to combine all types of resources, including supercomputers and clusters of machines, into a resource more powerful than any single resource. NetSolve [1] uses Remote Procedure Call (RPC) to harness resources distributed by ownership and geography. Condor-G [4] is targeted at utilizing the resources available from different institutions to speed the processing of simulations, large-scale optimization, and image processing among other computationally intensive tasks. Condor-G combines the multi-domain advantages of the Globus Toolkit with the management of resources available from the Condor System. An example of grid computing in use is Berkeley Space Sciences Laboratory's SETI@home project (Search for Extra Terrestrial Intelligence)[8], which uses Grid computing to harness CPU cycles used to analyze radio waves from space. SETI@home distributes sections of the sky to individual users to be analyzed and it runs the analysis as a screen saver on their idle PCs. When the user next logs onto the Internet the results are uploaded to the SETI@home site without any burden on the user.

2.3 Review of Existing REMOTE tool

REMOTE [2] is designed to be a lightweight tool that enables automatic remote execution of programs on Windows machines. It is designed to speed parallel independent replications of CSIM simulations and requires no configuration of the remote computers. REMOTE requires only a single program, small enough to be distributed via e-mail, to be run on the remote hosts. The REMOTE program then distributes input, output, and executable files from a single master to a list of remote machines. At the end of execution the output files are then transferred back to the master computer. All communications in REMOTE are implemented using the Winsock interface. The REMOTE tool is complex and has some known timing-related bugs.

3. GOAL OF REMOTE DISTRIBUTION

Figure 1 shows a remote distribution system. A single master machine distributes executables and, if applicable, their input and output files to remote PCs. The processes are executed on the remote machines and, at the completion of execution; the output is on the master machine. The fundamental goal of remote distribution of processes is to reduce execution time by harnessing the idle, non-dedicated, CPU cycles of network connected PCs. Remote distribution will, typically, be run during evenings and weekends when idle CPU time is maximized and users are unlikely to return to their machines quickly. Failures on these unmonitored machines can result in significant time setbacks. Thus, remote distribution programs must be very stable and not prone to failures. A remote distribution program must also insure that the overall execution time of distributed processes is significantly shorter than if executed on a single machine. Requirements for successful remote distribution of programs are:

1. The remote distribution program must be simple to allow for easy maintenance and modification.
2. The executables must be stand-alone processes that require no modification to enable remote execution.
3. Execution of programs must be automatic and not require any manual interaction.
4. Output files must be available on the master PC at the completion of execution.
5. There can be only one distributed process running at each remote host at any time.
6. Once a job completes, the next job is sent to the available host until all jobs are executed.
7. The time to assign and distribute a process must be significantly shorter than its execution time to ensure an overall decrease in execution time of the processes.
8. The failure of a job must be detected. It must not stop the distribution of the remaining jobs.
9. The failure of a host must be detected and the job reassigned to another host. Future jobs should not be assigned to the failed host.
10. Status and error messages must be displayed at the master PC to allow diagnosis of run-time errors.
11. A log file should indicate which jobs failed to execute and which hosts were invalid.

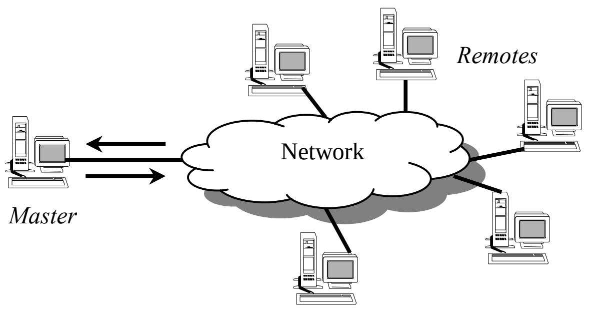


Figure 1. Master distributes jobs to remote hosts. Remotes execute the jobs. The output is transferred to the Master at the end of execution.

4. REMOTE++ DESIGN/IMPLEMENT

REMOTE++ implements time based parallelization of programs through Parallel Independent Replications. It is, therefore, a remote distribution program designed to meet the requirements presented in section 3. The REMOTE++ executable is built from the following files.

- remotepp.h – defines constants and prototypes

- remotepp.c – main program which launches the run, transfer and help functions
- run.c (syntax: remotepp run)– runs the jobs in joblist.txt on the hosts in hostlist.txt
- transfer.c (syntax: remotepp transfer *filename hostname*) – transfers the file *filename* from the master to host *hostname.*
- help.c (syntax: remote help) – displays the REMOTE++ help screen.

REMOTE++'s run function is a script that runs standard remote shell (rsh) and remote copy (rcp) commands within Windows threads to enable Parallel Independent Replications of processes. The flowchart in Figure 4 illustrates the structure of the run function. For each job in the joblist.txt file, the input file, output file, and executable file are opened to verify their existence. Each job with all three files valid will be placed in the job queue. It will also be assigned a host if there is a host in the hostlist.txt that has not been assigned to another job. The host will then be assigned to the host queue. If there are unassigned hosts in hostlist.txt after all valid jobs have been assigned to the queue, the hosts will be added to the host queue. For each job in the job queue that is ready to be run, a run thread will be started to allow execution of the job. If there is a job that has not been assigned a host and a host becomes available, that host will be reassigned to the job and a thread will be started. Once all threads have completed the run function will end. Figure 4 also diagrams the run thread of REMOTE++. Each thread will receive one job from the job queue. First the executable will be remote copied (with an rcp command) to the assigned host. If the rcp is successful, then the host is valid and the execution of the job will continue. , Otherwise, the host will be marked invalid and the job will be reassigned to the next available host. If the host is valid, the input/output method of the job will be determined. If the input/output method is "std", then a remote shell (rsh) command is executed and the job is run on the remote host with the input and output redirected from the master PC. If the input/output method is "file", the input and output files are copied (again with an rcp command) to the remote machine and the job is then executed with the rsh command. After execution, the output file must then be copied back to the master machine. Before the thread ends, the host is marked available so that future jobs can be run on that host.

There are two methods of input and output for REMOTE++. The "file" method is used for processes that read input from and write output to a file. The "std" method is used for processes that read input from standard input and write output to standard output. To enable processes which read from standard input to be executed independently of interaction with a user, all input must be written into a file. Redirection can then be used for input into and output from these processes. Different command sequences must be used to allow execution of these two file methods. Figure 2, below, shows sample command sequence for each method. If the "file" method (Figure 2 (a)), is used, the executable file (exe_file), input file (input_file), and output file (output_file) must be transferred to the remote machine (host name remote_host). This is done with a series of three rcp commands. The job is then executed, using an rsh command. After execution, the output file must be transferred back to the master (host name master_name) with an rcp command. If the "std" method (Figure 2 (b)), is used, then only the executable is copied to the remote machine. Redirection of input and output from the master PC is then used in the rsh command. The "file" method has more overhead in transfers (four rcp commands) than does the "std" method (one rcp command), which uses files stored on the master machine.

```
(a)  rcp –b /Remotepp/exe_file remote_name:/temp/
     rcp –b /Remotepp/input_file remote_name:/temp/
     rcp –b /Remotepp/output_file master_name:/temp/
     rsh remote_name c:/temp/exe_file
     rcp –b /temp/output_file  master_name:/Remote/

(b)  rcp –b /Remotepp/exe_file remote_name:/temp/
     rsh host_name c:/temp/exe_file <input_file
          >output_file
```

Figure 2. Sample rsh and rcp command sequence for "file" method (a) and "std" method (b).

4.1 User View of REMOTE++

REMOTE++ is run from the Windows console. Very little configuration is needed to enable the REMOTE++ program to run. Each remote host must be a network-connected machine running Windows 9x or higher. There must be an rsh and rcp daemon running on each remote host. A c:/temp/ directory must also exist on each remote host. Each executable, input and output file copied to the remote host will be saved in the c:/temp/ directory to allow easy execution and "clean up" of files on the remote machines. A c:/remotepp/ directory must exist on the master machine. The REMOTE++ executable, joblist.txt, hostlist.txt, and status.txt must be contained in this directory. Additionally, all executable, input, and output files listed in the joblist.txt should be located in the c:/remotepp/ directory. To execute the jobs in the joblist.txt on the hosts in the hostlist.txt, the command remotepp run must be executed at the command line on the master machine. A sample execution is shown in Figure 5. Figure 3 (a) and (b) show a sample joblist.txt and hostlist.txt, respectively. At the completion of execution, the status.txt file shows whether each file (executable, input, and output) for each job was found and the validity of each host. This information can be used to determine which jobs were completed. A sample status.txt is shown in Figure 3 (c).

```
(a)  file mm1.exe input1.txt output1.txt
     file mm1.exe input2.txt output2.txt
     file mm1.exe input3.txt output3.txt

(b)  giga2.csee.usf.edu
     giga3.csee.usf.edu

(c)  Mode is classic.
     Executable file mm1.exe found
     Input file input1.txt found
     Output file output1.txt found
     Mode is classic.
     Executable file mm1.exe found
     Input file input.txt found
     Output file output.txt found
     Mode is classic.
     Input file input.txt found
     Output file output.txt found
     giga2.csee.usf.edu is a valid host
     giga3.csee.usf.edu is a valid host
```

Figure 3. (a) Sample joblist.txt file (b) Sample hostlist.txt file (c) Sample status.txt file

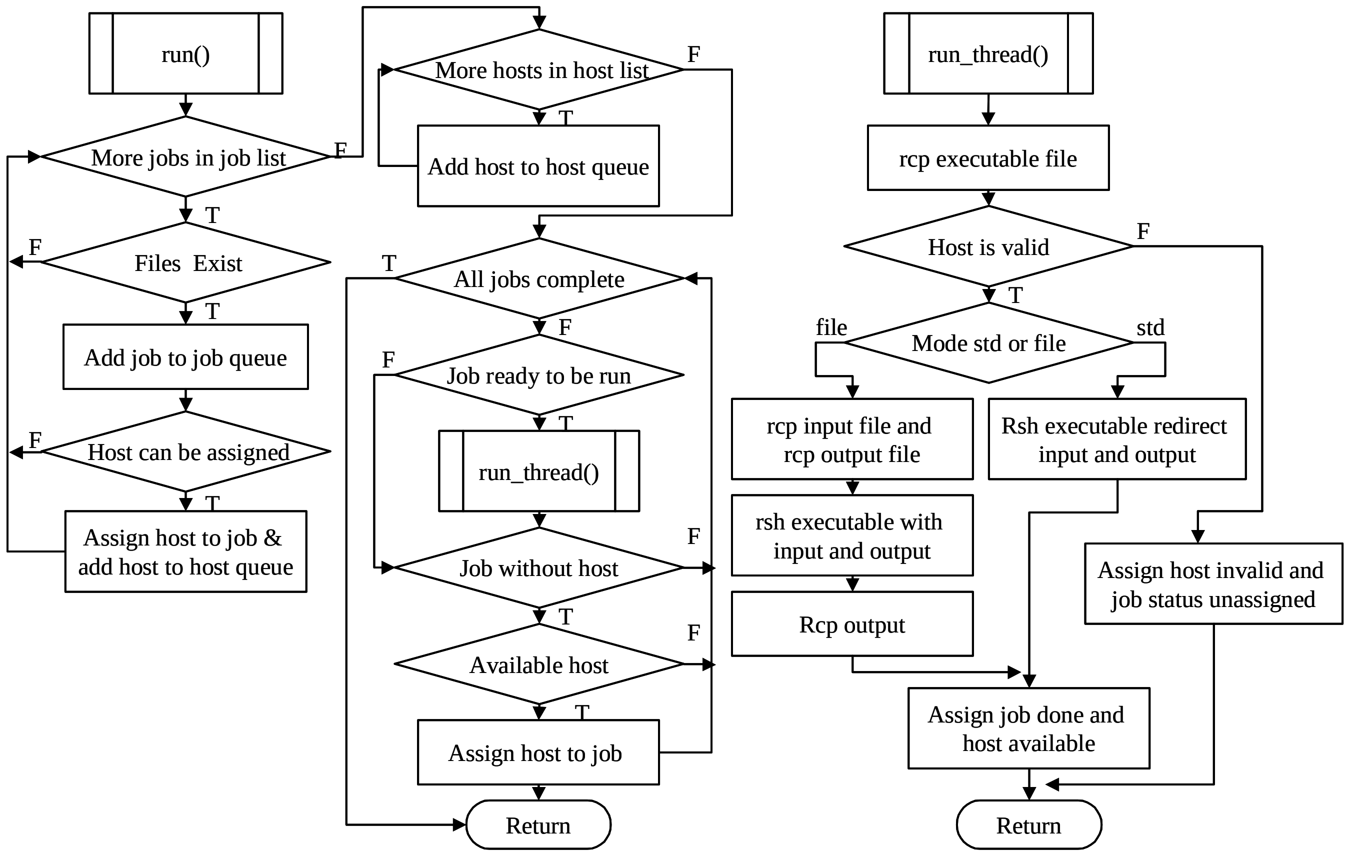
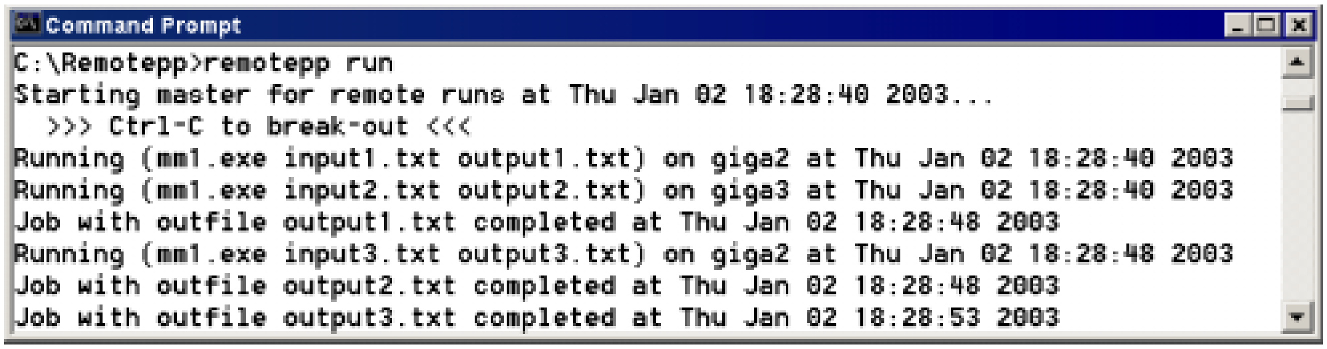


Figure 4. Flowchart of run function in REMOTE++



```
C:\Remotepp>remotepp run
Starting master for remote runs at Thu Jan 02 18:28:40 2003...
  >>> Ctrl-C to break-out <<<
Running (mm1.exe input1.txt output1.txt) on giga2 at Thu Jan 02 18:28:40 2003
Running (mm1.exe input2.txt output2.txt) on giga3 at Thu Jan 02 18:28:40 2003
Job with outfile output1.txt completed at Thu Jan 02 18:28:48 2003
Running (mm1.exe input3.txt output3.txt) on giga2 at Thu Jan 02 18:28:48 2003
Job with outfile output2.txt completed at Thu Jan 02 18:28:48 2003
Job with outfile output3.txt completed at Thu Jan 02 18:28:53 2003
```

Figure 5. Console window for "remotepp run" command for job list of Figure 2a host list of Figure 2b and status of Figure 2c

5.0 EVALUATION OF REMOTE++

The performance of queueing systems is often studied using simulation methods. A queueing system consists of a population of potential customers (which can be unlimited if not restricted to a certain group). These customers arrive in the queue and wait for service. The order in which the customers are served is determined by the queueing discipline or rules that determine where customers are inserted into and served from the queue. Once served, the customers are then returned to the population of possible customers. To gather statistical information these simulations must be executed numerous times with varying input. This makes such queuing simulations ideal for PIR, which will allow the simulations to be executed with multiple parameters on a pool of machines.

5.1 M/M/1 Queuing Systems

A M/M/1 queue is a specific queueing system. Figure 6 shows an M/M/1 queue. The specific requirements for an M/M/1 queue are:

1) Exponential inter-arrival of customers into the queue
2) A single exponential server
3) First In First Out (FIFO) queueing discipline
4) An unlimited queue capacity
5) Unlimited customer population

There are several variables that control simulation of an M/M/1 queue. The length of an M/M/1 queue is the number of jobs waiting to be serviced. At any time this length (L) can be determined by the arrival rate of items into the queue (λ) and the wait time in the queue (W). This relationship is $L = \lambda W$. The queue length can also be simply calculated from the utilization of the queue by $L = \rho/(1-\rho)$. The utilization of the queue is the ratio of the arrival rate of items into the queue and the service rate of items leaving the queue. As the utilization of the queue increases and approaches 100%, the length of the queue approaches infinity.

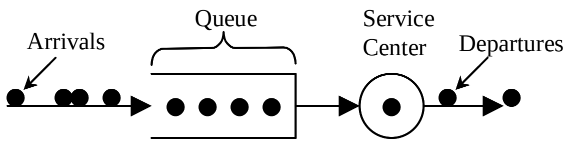


Figure 6. M/M/1 queue

5.2 Evaluation of REMOTE++

A relationship exists between the utilization of an M/M/1 queue, the length of that queue, and the simulation time. To determine this relationship, the M/M/1 simulation must be executed numerous times with utilizations approaching 100% and the related simulation times calculated. No output from one simulation is needed for any other simulation to complete. Therefore, each execution is independent. REMOTE++ was used to determine the relationship between the M/M/1 queue utilization and the simulation run time for mean queue length within a percent of the theoretical length. REMOTE++ was used to distribute simulations with increasing utilization to a pool of 5 remote machines. Three of these were Pentium III 866MHz machines and two were Pentium III 700 MHz machines. The M/M/1 simulation reads the target utilization and the desired margin of error (i.e., from the known theoretical length of the queue as calculated by $L = \rho/(1-\rho)$) as input from a file. A 10% margin of error was used for this evaluation. Utilization was started at 1% and increased to 99.5%. Evaluation of each utilization was executed several times using different seeds and the results were then averaged to gather statistical results.

Using REMOTE++ to run these simulations on this pool of machines took about 20 minutes. If executed on a single 866 Mhz machine, this would have taken approximately 50 minutes. Therefore, it executed about two and a half times faster on five machines than it would have on one machine. Since the simulations are independent, the overall increase was expected to be about five times faster when executed on five machines (about 10 minutes). The plot shown in figure 7, diagrams these execution times. The difference in actual and projected execution time is due to overhead in distributing the simulations. There is about seven seconds of overhead in executing each job in the job list, which has input/output method "file". Since many of the simulations in the job list had low target utilizations, they took only seconds to execute. Therefore, this seven seconds of overhead was significant in the time needed to complete these simulations. If each of the jobs in the job list took longer to execute, the overhead would be offset by the execution time. The speed up in this case would have been closer to five times.

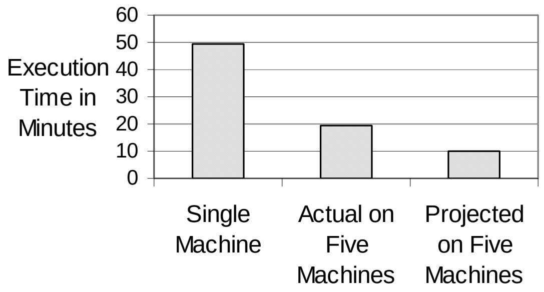


Figure 7. Execution times of M/M/1 simulation. Column 1 is actual execution time when executed one 866 MHz Pentium III computer. Column 2 is the actual execution time when executed on five machines (three 866MHz and two 700 MHz Pentium III computers). Column 3 is the projected execution time on five machines based on a 5 times speed up.

The plot shown in figure 8, shows the results gathered in the M/M/1 queue simulation run with the REMOTE++ program. It illustrates the relationship between the target utilization and the simulation time for the M/M/1 queue for a target utilization between 90% and 99.5% and a margin of error of 10%. From these results one can conclude that as the target utilization approaches 100% the simulation time of the M/M/1 queue increasingly grows longer.

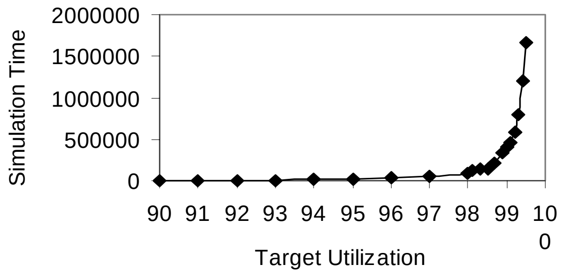


Figure 8. Simulations time versus target utilizations for an M/M/1 queue with margin of error of 10.0%.

6.0 SUMMARY AND FUTURE WORK

The REMOTE tool was designed to harness the non-dedicated idle CPU cycles of Windows PCs through process distribution and remote execution. REMOTE++ improves upon this tool. There is little difference in the remote distribution overhead between REMOTE and REMTOE++. However, REMOTE++ eliminates the complex Winsock interface used by REMOTE and replaces it with the standard rsh (remote shell) and rcp (remote copy) commands. Both REMOTE and REMOTE++ are relatively small programs consisting of 1100 and 400 lines of standard C respectively. Therefore it is not the reduction in code length that makes the REMOTE++ code an improvement over the original REMOTE tool. The standard rsh and rcp commands implemented in REMOTE++ make it much less cumbersome. It therefore, enables easy maintenance and modification of the tool. In addition to executing programs, which read from and write to files as REMOTE was designed to achieve, REMOTE++ can execute programs, which read from standard input and write to standard output. REMOTE++ only requires that a single program (remotepp.exe) be executed on the master machine to distribute and execute the jobs from the job list to the hosts in the host list. The remote machines need only run an rsh/rcp daemon to enable transfer and execution of programs. No REMOTE++ program need be run on the remote machines. REMOTE++ will skip a job if any portion of it is invalid (including executable, input or output). Jobs assigned to an invalid host will be reassigned to the next available host and no future jobs will be assigned to the invalid host. The status.txt log will list the validity of each file and host at the completion of execution. The REMOTE++ program and the M/M/1 queue simulation model used to evaluate it are freely available from the author as open source with no restrictions on use.

There are several known problems with REMOTE++ that are the subject of future work. There is no free rsh/rcp daemon available that can reliably support execution of the REMOTE++ program. The rsh/rcp daemons available from independent vendors cost approximately $40.00, which may limit use of the REMOTE++ program. A reliable, free daemon, therefore, needs to be developed to maximize the use of this program. REMOTE++ also requires that the user list whether each executable reads and write to files or standard input and output. It also requires that both input and output be done with the same method. Future improvements on REMOTE++ should enable automatic detection of the proper method to be used and allow any combination of methods to be used. Security features should be implemented in REMOTE++ that are not dependant on the rsh/rcp daemon and do not require configurations on the remote PCs.

7.0 REFERENCES

[1] Arnold, D.C., and Dongarra, J. The Netsolve environment: progressing towards the seamless grid. International Workshop on Parallel Processing, (Proceedings 2000), 199-206.

[2] Christensen, K.J. REMOTE: A Tool for Automatic Remote Execution of CISM Simulation Models. Proceedings 35th Annual Simulation Symposium, (2002), 134-142.

[3] Condor for Microsoft Windows4.0. http://www.cs.wisc.edu/condor/manual/v6.1/5_Condor_Microsoft.html

[4] Frey, J., Tannenbaum, T. Livny, M., Foster, I., and Tuecke, S. Condor-G: a computation management agent for multi-institutional grids. Proceedings 10th IEEE International Symposium on High Performance Distributed Computing, (2001), 55-63.

[5] Litzxkow, M., Livny, M., and Mutka, M. Condor – A Hunter of Idle Workstations. Proceedings of the 8th International Conference on Distributed Computing Systems, (June 1988), 104-111.

[6] Mota, E., Wolisz, A., and Pawlikowski, K. Comparing overlapping batch means and standardized time series under Multiple Replications in Parallel. Simulation and Modeling. Enablers for a Better Quality of Life. 14th European Simulation. Multiconference (2000), 43-48.

[7] Yau, V., and Pawlikowski, K. AKAROA: A package for automating generation and process control of parallel stochastic simulation. Australian Computer Science Communications, Volume 15, Issue 1, Part A, (1993), 71-82.

[8] Young, E., and Cliff, P. Distributed Computing – the SETI@home Project. Ariadne Issue 27, (March 2001), http://www.ariadne.ac.uk/issue27/seti/