Using a Desktop Grid to Support Simulation Modelling

Navonil Mustafee and Simon J E Taylor
Centre for Applied Simulation Modelling
School of Information Systems, Computing and Mathematics
Brunel University
Uxbridge, Middlesex, UB8 3PH, UK.
navonil.mustafee@brunel.ac.uk, simon.taylor.brunel.ac.uk

Abstract. *Simulation is characterized by the need to run multiple sets of computationally intensive experiments. We argue that Grid computing can reduce the overall execution time of such experiments by tapping into the typically underutilized network of departmental desktop PCs, collectively known as desktop grids. Commercial-off-the-shelf simulation packages (CSPs) are used in industry to simulate models. To investigate if Grid computing can benefit simulation, this paper introduces our desktop grid, WinGrid, and discusses how this can be used to support the processing needs of CSPs. Results indicate a linear speed up and that Grid computing does indeed hold promise for simulation.*

Keywords. Grid, Desktop grid, WinGrid, Simulation experimentation, COTS Simulation Packages.

1. Introduction

Grid computing seeks to achieve the secured, controlled and flexible sharing of resources (for example, multiple computers, software and data) among various dynamically created virtual organizations[1][7]. These virtual organizations are generally setup for collaborative problem solving and access to grid resources are limited to those who are part of the project. The creation of an application that can benefit from Grid computing (faster execution speed, linking of geographically separated resources, interoperation of software, etc.) typically requires the installation of complex supporting software and an in-depth knowledge of how this complex supporting software works [10].

Simulation modelling is a field that can potentially benefit from Grid computing. Consider a model that takes ten minutes to run. If a modeller wishes to perform six experiments then the modeller will have to wait one hour plus setting up time for each experiment. If each of these experiments require ten replications, then the total experimentation time will be ten hours. Some models require much experimentation and some models take longer to run. The consequence of this is that for some simulation projects, experimentation can take a considerable length of time (or only be partially done). It appears that simulation modelling should be able to benefit from the processing power provided by Grid computing techniques.

We consider this from the viewpoint of the user of *COTS Simulation Packages* (CSPs). CSPs are visual interactive modelling software used by many practitioners in the practice of simulation modelling. These include Arena, Automod, Promodel, Simul8, Witness, etc. We use the term as a convenient way to refer to these packages. These CSPs are typically standalone packages that run on a single desktop PC on the Windows operating system. Users of CSPs tend to be skilled in simulation modelling and not computer science (as many users of Grid computing are). Vendors of CSPs consequently change the functionality of their CSPs on an incremental basis. Major possible changes to their packages are often prohibitively costly and do not have a guaranteed return on investment (ROI). Grid support for CSPs must therefore take into account that these packages are windows-based, their users are specialists in simulation modelling and not computing and any technological solution must be developed with little or no change to the CSP.

The paper is structured as follows. In section 2 we review the relevant current approaches to Grid computing and the notion of *Desktop Grids*.

[1] A group of individuals and/or institutions engaged in some joint task who share resources by following clearly stated sharing rules. These rules define what is shared, who is allowed to share and the conditions under which sharing occurs.

Section 3 introduces our Desktop Grid system called *WinGrid*. Section 4 discusses how we used WinGrid to "Grid-enable" the CSP Simul8 and a simulation application. Section 5 presents the results of this and shows the speed up that we obtained using a small number of desktop PCs. Section 6 considers the implication of this and draws the paper to a close. Note that we are aware of some CSPs that have specialist experimentation tools that make use of multiple computers. However, this research considers how CSPs might make use of such resources via the use of generally available Grid computing software targeted at desktop users.

2. Desktop Grids

While much of Grid computing is focussed on meeting the needs of large virtual organizations, *Desktop Grid Computing* or *Desktop Grids* addresses the potential of harvesting the idle computing resources of desktop PCs [4]. These resources can be part of the same local area network (LAN) or can be geographically dispersed and connected via a wide area network such as the Internet. Studies have shown that desktop PCs can be under utilized by as much as 75% of the time [12]. Given the number of desktop computers across the world, this represents an enormous computing resource. The immediate implication of this is that software applications can potentially run substantially faster. In enterprises, this also means that the ROI of enterprise computing resources can also be potentially increased.

Two principal types of desktop grids have emerged. These are *Public Resource Computing* and *Enterprise Desktop Grid Computing*. Both these are based on variants of the master/workers distributed computing architecture [3]. In such a model a user launches an application on a master computer that is responsible for allotting work generated by the application to the available worker computers for processing. The individual results are returned by the workers to the master for compilation by the application and presentation to the user.

2.1 Public Resource Computing

Public-resource computing (PRC) refers to the utilization of desktop grids comprising millions of desktop computers primarily to do scientific research [1]. Berkeley Open Infrastructure for Network Computing (BOINC) [19] is the most widely used desktop grid application that supports scientific projects with diverse objectives such as searching for evidence of extraterrestrial intelligence, studying climate change, improvement in the design of particle accelerators, finding cures for human diseases and searching for gravitational waves from space. Non-BOINC based projects use their own software to facilitate research with similar objectives, for example, finding a cure to cancer [14], understanding protein folding [13] and computing mersenne prime numbers [8]. The participants of PRC projects are volunteers who register with one or more such projects and install the required desktop grid software. This software then contacts the central project servers and downloads work units for processing (in case of BOINC it also downloads project specific executable code as BOINC is a general purpose PRC client). The time it takes to complete the execution of a work unit and return back the result depends, among other things, on the machine hardware, the amount of time a PC is left running and user preferences. The volunteers are themselves unable to use the underlying desktop grid infrastructure, of which they themselves are part of, to perform their own computations.

2.2 Enterprise-wide Desktop Grid Computing

We use the term *Enterprise-wide Desktop Grid Computing* (EDGC) to refer to a grid infrastructure that is confined to an institutional boundary, where the spare processing capacity of an enterprise's desktop PCs are used to support the execution of the enterprise's applications. User participation in such a grid is not usually voluntary and is governed by enterprise policy. Applications like CONDOR [11], Platform LSF [15], DCGrid [6] and GridMP [18] are all examples of EDGC. Unlike the PRC model these applications usually allow users to submit jobs for processing.

2.3 Desktop Grids and CSPs

How can a desktop grid support the needs of CSP experimentation? To recap, our aim is to create a system that takes into account that these packages are windows-based, their users are specialists in simulation modelling and not

computing and any technological solution must be developed with little or no change to the CSP.

Building on PRC and EDGC, one possibility is to "bundle" the CSP along with each desktop grid worker. Thus, whenever a desktop grid worker is started the CSP is also loaded. In an enterprise-wide desktop grid the worker usually runs in a "sandbox". We call this sandbox the Desktop Grid Virtual Machine (DGVM) and this provides logically separate, secure execution environment for both the host and guest processes.

In DCGrid for example, the DGVM is called the Entropia Virtual Machine (EVM) and it wraps interpreters like cmd.exe, perl and Java Virtual Machine to prevent unauthorized access to a computer [2]. Thus, it might be possible to include a CSP installation inside the EVM and offer it as part of an Entropia installation. In this case the master will need to send the data files associated with the simulation and a script file to trigger the CSP execution in the worker DGVM. The simulation results would be collected in a file, which would then be sent back to the master. The problem with this approach is that it would require major changes to the CSP (integrating CSP into a DGVM).

An alternative solution would be to install the CSP in the worker nodes as a normal application and then have the master communicate directly with that application. The drawback with this is that the sandbox security mechanism which is present in most EDGC approaches would have to be forfeited. However, as simulations are created by *trusted* employees running *trusted* software within the bounds of a firewalled network, security in this open access scheme could be argued as being irrelevant (i.e. if it were an issue then it is an issue with the wider security system and not the desktop grid).

Let us now consider our approach to supporting simulation with desktop grids by introducing our *WinGrid*.

3. WinGrid: A Desktop Grid for Windows

WinGrid is a distributed middleware application written in Java that is based on the desktop grid master/workers architecture and is shown in Fig. 1. As can be seen, WinGrid consists of four different parts: the *manager application* (MA), the *WinGrid Job Dispatcher* (WJD), the *worker application* (WA) and the *WinGrid Thin Client* (WTC). The MA runs on the manager computer (the application user's computer) and is software written specifically for the management of the application running over the desktop grid (in our case study this is Excel). The MA interacts with the WJD also running on the master computer and passes work to, and receives results from, the WJD. The WAs and WTCs run on each worker computer. The WJD sends and receives work to and from the WTCs. The WTCs in turn send and receive work to and from their WA. The WAs are unmodified application software connected via a COM interface with the WTCs. The WTC is also responsible for advertising and monitoring local resources, accepting new jobs from the master process and returning back the results, and provides an interface through which the desktop user can set his preferences (when guest jobs are to be run, applications to share etc.). As seen in Fig. 1 below, the user submits a job through the MA (1), which in turn interacts with the WJD process (2) in the manager computer to send work (4) to the WinGrid workers and their WTCs (3). The WTC pass this work to their WA for processing (5) and returns the result to the WJD (6). The results of all the sub jobs are communicated back to the MA which then collates the results and presents it to the user.

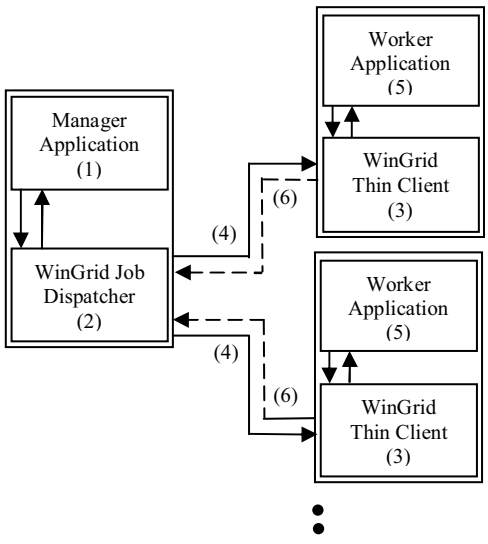

Figure 1. WinGrid nodes running a user job

4. Case Study: Simul8-WinGrid Integration

For a desktop grid to support experimentation with a CSP it must be able to access the CSP and we require (1) CSP accessibility through desktop grids software, (2) an interface which allows us to access the internal functionality of a CSP, for example, starting and stopping a simulation, changing parameter values, retrieving results etc., (3) an interface to desktop grid which allows us to receive and send information through the network, (4) a mechanism which interacts with both the CSP and desktop grid interface, (5) a standard way to represent the batch simulation experiments which would be submitted to the desktop grids, and (6) a mechanism which can read the job submission file, extract parameters of each simulation and collectively present the results. Of these, (2) and (3) are provided by most CSPs and desktop grids respectively. (1, 4, 5, 6) will need some implementation specific to the CSP and the desktop grid in question.

Simul8 is a discrete-event CSP that enables users to rapidly construct accurate, flexible and robust simulations using an easy-to-use visual modelling interface [5]. In order to grid-enable Simul8 we have integrated it with WinGrid using the Component Object Model (COM). COM is a Microsoft technology that allows different software components to communicate with each other by means of interfaces [9]. Simul8 provides a Windows COM interface that can be used from within any COM-compliant language to "drive" Simul8 [16].

Each simulation model is different and we need to make only model-related COM calls. A custom built Simul8 adapter wraps the code necessary to interact with the model in question. This adapter (code) will be used by each WinGrid client along with the simulation file (data) to interact with Simul8 and perform the required experiments. In the actual implementation the WinGrid job dispatcher does not explicitly transport code and data over the network because each WinGrid client has access to a shared drive (Windows OS takes care of this implicitly).

For the purpose of this paper we define a *Simul8 job* as a collection of experiments performed on a Simul8 model. The experiment parameters are entered by the user through Microsoft Excel spreadsheet (Fig. 2). We choose Excel because it has well documented COM interface which allows us to easily integrate it with WinGrid and is a technology familiar with simulation modellers. The user submits the Simul8 job (Excel spreadsheet) by invoking the job dispatcher, which in turn reads the file and distributes the experiments between the nodes. As soon as each result is reported back to the job dispatcher it is conveyed to the user through the Excel interface.

	A	B	C	D	E	F	G	H	I	J
2	Simul8 Experiment Controller (Desktop Grid Version)									
3	Exp No	Max Circling	Max Arrival Rate (hour)	Planes Circling	Planes On Approach	Planes At Terminal	Planes Waiting for Clearance	Planes Landed and Departed	Planes Diverted	Processed By
4	1	33	22	27	27	34	2	1187	4	192.168.0.210
5	2	38	26	38	21	34	7	1274	12	192.168.0.211
6	3	15	50	15	20	34	15	1305	9	192.168.0.212
7	4	25	54	24	28	34	13	1304	21	192.168.0.213
8	5	36	54	36	32	33	4	1303	22	192.168.0.210
9	6	27	56	27	26	34	11	1304	21	192.168.0.212
10	7	35	45	35	22	33	2	1302	17	192.168.0.211
11	8	22	28	21	20	34	13	1288	3	192.168.0.213
12	9	35	49	33	40	34	1	1304	18	192.168.0.213
13	10	18	33	18	15	34	2	1301	2	192.168.0.212
14	11	35	32	34	23	34	10	1290	10	192.168.0.210
15	12	28	59	28	25	34	15	1304	19	192.168.0.211
16	13	31	24	30	12	33	5	1270	2	192.168.0.212
17	14	14	20	12	7	28	0	1107	1	192.168.0.213
18	15	36	39	36	25	34	9	1296	21	192.168.0.210
19	16	15	52	15	20	34	14	1304	8	192.168.0.211
20	17	12	50	12	22	34	15	1304	5	192.168.0.213
21	18	37	23	37	23	33	2	1232	7	192.168.0.212
22	19	31	35	29	18	34	9	1295	8	192.168.0.211
23	20	25	55	24	32	34	14	1304	18	192.168.0.210
24	21	38	33	38	19	34	13	1290	9	192.168.0.213
25	22	18	48	18	9	34	14	1303	8	192.168.0.212

Figure 2. Spreadsheet showing input parameters (Cols. A-C) , results (Cols. D-I) and the WinGrid node which executed the simulation (Col. J)

Since WinGrid is written Java (a non-COM compliant language), we have used Java Native Interface technology [17] for communication between Excel Adapter, WinGrid and the Simul8 Adapter. Fig. 3 shows the architecture of the CSP and WinGrid.

5. Experiments and Results

To demonstrate the potential of achieving a speedup when using CSP over WinGrid, we have used the airport simulation demonstration that comes as part of the standard Simul8 distribution to conduct performance tests. This model accepts input parameters such as max number of planes allowed to circle the airport and the maximum arrival rate per hour. The performance was measured in terms of the time taken to execute 50, 100, 150 and 200 experimentations of the model. In both standalone and 4-node WinGrid environments we used the same parameter values. An Excel spreadsheet similar to the one shown in Fig. 2 was used to automate the

running of the standalone simulation. Each experiment was run to a preset simulation time.

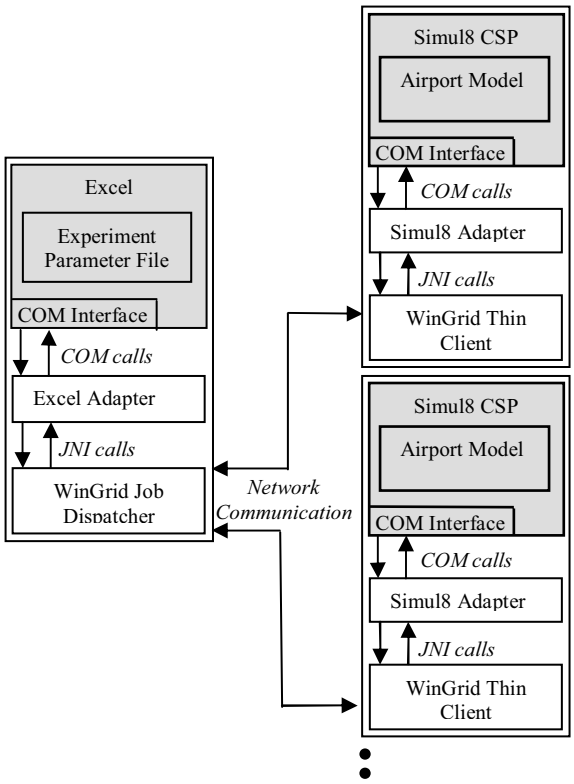

Figure 3. Interactions between WinGrid nodes during simulation run

All experiments were performed on Dell Inspiron laptops running Microsoft Windows XP OS with 1.73GHz processors and 1GB memory, connected through a 100Mbps CISCO switch. The results obtained are shown below.

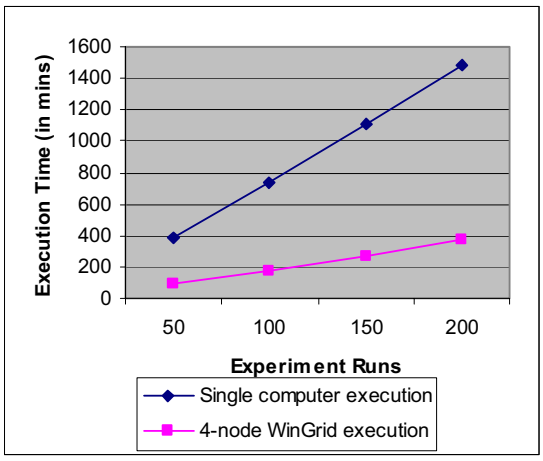

Figure 4. Total time taken to execute simulation experiments

As can be seen in Fig. 4, the 4-node WinGrid was able to complete all its experiments approximately 4 times faster when compared to standalone execution. Note that this assumes that all four computers are dedicated to this task. In a "real" office environment this might not be the case. However, with careful management this speedup is not at all unrealistic.

6. Conclusion

Studies have shown that desktop PCs remain vastly underutilized in an organizational setting and desktop grids give us an opportunity to utilize these otherwise untapped resources to perform compute intensive and repetitive jobs. Simulation modelling stands to benefit through adoption of this technology. Simulations in industry are typically created using a CSP and any desktop grid solution which supports a CSP is expected to appeal to them. WinGrid, unlike other desktop grids, provides architecture for application level CSP integration with the underlying grid infrastructure. WinGrid therefore seeks to reduce the time taken to simulation experimentation by sharing the processing load across many desktop computers.

Future work in this area will involve WinGrid integration with other Windows applications (we have also successfully run Monte Carlo simulations in multiple instances of Excel spreadsheets following a similar approach). We plan to incorporate features like user activity detection and check pointing onto the WTC and job migration, fault tolerance, workflows into the WJD. Some of these features will help us to make WinGrid a testbed for running CSP based distributed simulations.

Acknowledgements

The authors would like to thank Dr. Mark Elder (founder and CEO of SIMUL8 Corporation) for providing the Simul8 licenses and their generous on-going support. Part of this work was funded by the WestFocus GridAlliance ICT programme.

References

[1] Anderson D.P. BOINC: a system for public-resource computing and storage. In: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing; 2004 November; 2004.p.4-10.

[2] Calder B, Chien A.A, Wang J, Yang D. The entropia virtual machine for desktop grids. In: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments; 2005 June 11-12; Chicago, IL, USA ; 2005.p.186-196.

[3] Chakravarti A.J, Baumgartner G, Lauria M. Application-specific scheduling for the organic grid. In: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing; 2004 November; 2004.p.146-155.

[4] Choi S, Baik M, Hwang C, Gil J, Yu H. Volunteer Availability based Fault Tolerant Scheduling Mechanism in Desktop Grid Computing Environment. In: Proceedings of the 3rd IEEE International Symposium on Network Computing and Applications; 2004 August; 2004.p. 366-371.

[5] Concannon K.H, Hunter K.I, Tremble J.M. SIMUL8-planner simulation-based planning and scheduling. In: Proceedings of the 35th conference on Winter simulation; 2003 December; New Orleans, Louisiana; 2003.p.1488-1493.

[6] Entropia Inc; 2006. http://www.entropia.com/ [02/01/2006].

[7] Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications 2001; 15(3):200-222.

[8] George Woltman. GIMPS; 2006. http://www.mersenne.org [02/06/2006].

[9] Gray D.N, Hotchkiss J, LaForge S, Shalit A, Weinberg T. Modern languages and Microsoft's component object model. Communications ACM 1998; 41(5):55-65.

[10] Jaesun H, Daeyeon P. A lightweight personal grid using a supernode network. In: Proceedings of the 3rd International Conference on Peer-to-Peer Computing; 2003 Sept 1-3; 2003.p. 168-175.

[11] Litzkow M, Livny M, Mutka M. Condor - A Hunter of Idle Workstations. In: Proceedings of the 8th International Conference of Distributed Computing Systems; 1988 June. 1988.p.104-111.

[12] Mutka M.W . Estimating capacity for sharing in a privately owned workstation environment. IEEE Transactions on Software Engineering 1992; 18(4):319-328.

[13] Pande V , Stanford University. Folding@Home; 2006. http://folding. stanford.edu/ [02/06/2006].

[14] Parabon Computation Inc. Compute Against Cancer; 2006. http://www.computeagainst cancer.org/ [02/06/2006] .

[15] Platform Computing; 2006. http://www.platform.com/ [02/06/2006].

[16] Simul8 Corporation. Simul8: Manual and Simulation Guide. 2003.

[17] Sun Microsystems Limited. Java Native Interface (2003). http:/java.sun.com/j2se/1.4.2/docs/guide/jni [02/06/2006] .

[18] United Devices Inc; 2006. http://www.ud. com/ [02/01/2006].

[19] University of California. Berkeley Open Infrastructure for Network Computing; 2006. http://boinc.berkeley.edu/ [02/06/2006] .