Journal of Simulation (2011) 5, 266–278 © 2011 Operational Research Society Ltd. All rights reserved. 1747-7778/11
www.palgrave-journals.com/jos/

High performance spreadsheet simulation on a desktop grid

J Pichitlamken[1]*, S Kajkamhaeng[2], P Uthayopas[2] and R Kaewpuang[2]

[1]Kasetsart University, Bangkok, Thailand; and [2]High Performance Computing and Networking Center, Kasetsart University, Bangkok, Thailand

In this paper, a high performance spreadsheet simulation system called S3 is presented. Our approach is to add power of parallel computing on Windows-based desktop grid into popular Excel models by using standard Web Services and Service-Oriented Architecture. The complexity of parallelism can be hidden from users through a well-defined computation template. Our desktop grid is built from off-the-shelf office PCs connected together using a high-speed network. The experimental results show that the prototype system can deliver high performance. We can obtain more than seven times speedup for some test applications on a 8-PC system. Moreover, the implementation is highly scalable since 80–95% parallel computing efficiency can be maintained as the problem size grows.

Journal of Simulation (2011) **5,** 266–278. doi:10.1057/jos.2010.23; published online 10 December 2010

Keywords: spreadsheet simulation; desktop grid; parallel simulation

1. Introduction

Spreadsheet is a widely used computer application because of its versatility and ease of use in calculation and modelling a problem. Currently, spreadsheet becomes an important tool for modelling, simulation, and problem solving. For example, stochastic simulation on spreadsheet is applied to many types of problems; such as, financial risk analysis (Paisittanand and Olson, 2006) and operational risk analysis (Shariff *et al*, 2006). Luce *et al* (2005) implement their product development model of a liquid packaging pump on Microsoft Excel with Crystal Ball add-in (www.oracle.com/crystalball) to simulate and determine optimal design parameters (eg, stroke length or different motor options) that minimizes unit production cost.

Many textbooks use Excel as a computational tool due to its ease of use and flexibility. For example, Ragsdale (2004) and Stevenson and Ozgur (2007) are introduction to operations research (OR); Myerson (2004) discusses about decision analysis models; and Seila *et al* (2003) cover spreadsheet simulation and discrete-event simulation (DES). Examples of DES models that can be built on spreadsheets are simple queues, discrete-time Markov chain models, regenerative processes, and inventory models (see Chapter 4 of Seila *et al*, 2003 and Chapter 2 of Banks *et al*, 2010 for more details). Spreadsheet simulation is also a teaching aid on the subjects of supply chain management (Adams *et al*, 2005), queues (de Mesquita and Hernandez, 2005), and decision analysis (Ståhl, 2005). See Seila (2006) for a short tutorial on spreadsheet simulation.

Spreadsheet is also chosen as a user's interface for optimization modelling. Because of the availability of Excel's Solver add-in, modelling a linear or integer programming problem on spreadsheet is familiar to many OR practitioners. Moreover, Excel can be customized through user-defined Visual Basics for Application (VBA) macros or add-ins that allow users to include their own functions. Many of these spreadsheet-based models consider real-world problems: Sakalli and Birgoren (2009) formulate a blending problem for a brass casting plant. The objective is to determine the optimal mix of raw materials (20–30 types) that minimizes total cost and satisfies product specifications and required quantity (70–80 constraints). Though this single-blend model is not large, it is solved 20–30 times a day, whenever a brass type is changed, and it is put into actual use. The larger multi-blend model is also presented, but it is under development.

Cunha and Mutarelli (2007) determine the best policy for producing and distributing weekly news magazines in Brazil. Their mixed-integer linear model consists of over 10 000 decision variables (DVs) and 1000 constraints. It is formulated in Excel and then optimized with a Lindo's product called What's Best! Similar to Cunha and Mutarelli (2007), LeBlanc and Galbreth (2007) also discuss implementation issues of building a large optimization model in Excel; a provided example of a large model is a supply-chain system

Correspondence: J Pichitlamken, Department of Industrial Engineering, Faculty of Engineering, Kasetsart University, Bangkok, 10900, Thailand.
E-mail: juta.p@ku.ac.th

with '39 000 rows and more than 50 megabytes in size'. Their approach is to include VBA macros and to intelligently use built-in Excel functions (eg, SUMIF) to make Excel models more compact and to increase its functionalities.

When a problem size is large, computation power of a single computer might not be enough; users may have to wait for a few hours to see simulation results. Thus, accelerating computing speed for Excel in a transparent way is beneficial since it allows users to quickly 'play' with the problem and gain insights. One approach is to enhance Excel using a parallel and distributed computing.

A recent survey on the future of DES reports that although parallel and distributed simulation is promising, it is still used only by specialists because simulation applications are difficult to develop (Taylor and Robinson, 2006). Boer *et al* (2009) present survey results on the use of distributed simulation in industry, specifically finding out why industry outside defence (eg, vendors of commercial-off-the-shelf simulation packages) does not use it although much research has been done in the area. Some of the factors are cost, the use of high-level architecture for integrating simulation models, and reusability of models.

In this paper, we build a spreadsheet simulation system (**S3**) that is easy to use and computationally fast by utilizing power of parallel computing on a Windows-based desktop grid. The idea is to break a large simulation problem into many small sub-problems that can be executed concurrently on multiple computers. A middleware is developed to control the distribution and execution of these sub-problems. Microsoft Excel spreadsheet is enhanced with the VBA macro to seamlessly use the newly developed parallel computing capability to speed up its execution of a large simulation problem. When simulation runs are faster, analysts can execute a simulation model under multiple sets of DVs, or they may want to find the best set of DVs for the system of interest (the so-called optimization via simulation. See Andradóttir, 2006 for an overview).

S3 is designed for spreadsheet-based models where some input parameters are uncertain and/or DVs exist. In this paper, we illustrate our approach via two test problems. The first linear programming model is taken from LeBlanc and Galbreth (2007) where a chemical company, Applichem, produces a chemical from many plants in different countries for export and domestic markets. The overall costs at each plant depend on exchange rates, which are uncertain, and they can change the optimal export shipments. S3 samples the exchange rates multiple times, and for each sample path (realization), Excel's Solver is invoked to find the optimal shipment. Information on variability of total cost can provide measures of risk involved. Section 4.1 explains the problem in more details.

In the second example, we consider a project selection problem: The company wants to determine a subset of projects to invest such that the firm's expected profit is maximized (Ragsdale, 2004). Each project incurs an initial cost, and it may or may not be successful. If it is successful, its revenue is also random. Section 4.2 describes it in more details.

The novelty of our approach is how to seamlessly integrate parallelism with simulation runs by providing a 'template' as a guideline for quick development of spreadsheet simulation models. In this paper, we consider two key design problems:

1. To design spreadsheet simulation infrastructure, which is inexpensive and easy to maintain.
2. To design an Excel user's interface in such a way that users can slightly modify their existing models or build their models without being aware of parallelism and infrastructure behind. The parallel computation should be hidden from users as much as possible to make it user-friendly.

We address the first issue with a desktop grid that consists of commodity PCs connected together using a high-speed network. Grid computing focuses on large-scale resource sharing. Grid architecture specifies protocols that define 'the basic mechanisms by which users and resources negotiate, establish, manage, and exploit sharing relationships' (Foster *et al*, 2001).

Desktop grid computing generally consists of clients, workers, a manager, and servers. A client submits jobs that are executed by workers. A manager is responsible for job scheduling and resource management. Servers are used for data storage. A well-known example of a desktop grid is SETI@home, which is based on BOINC (Anderson, 2004). Desktop grids are appealing for they allow intensive computation to be performed at low cost. However, the main challenge is stability and security. Given that our desktop grid consists of PCs within the same organization, the security issue may not be too prohibitive.

Since we target on a desktop grid of PCs, a parametric approach is one way to harness the computing power in an environment where network communication latency is high. Although it is possible to utilize a non-parametric approach, the performance attainable will not be as good, and the scalability of the system may be unsatisfactory. In fact, a fine-grained parallelism inherent in a single spreadsheet model is better solved using multicore processors or GPUs, which involve only fast internal memory bus transfer. But the system scalability in terms of the number of processors and memory size may be limited by hardware constraints.

The partitioning of a single spreadsheet can be done by employing the parallelism inside a single sheet. The basic idea is to look into row and column parallelism and dependency, similar to the single instruction, multiple data (SIMD) computation. Typically, Excel evaluates formulas and displays the results as values in the cells that contain the formulas (Ecklund, 2007). By default, Excel has 'automatic calculation': it recalculates any cells that are dependent on other cells whose values have been changed. Abramson *et al* (2001) call it sequential calculation. This feature complicates evaluation of multiple cells that have built-in formulas in parallel.

As a result, the issue of hiding parallelism is resolved by separating random inputs and simulation outputs so that they are on separate Worksheets (A .xls file is an Excel Workbook which consists of Worksheets). We have a Worksheet template where a user specifies the number of inputs and outputs and their respective cell locations. When simulation runs are finished, users get outputs of each replication (if there is only one set of DVs) or sample means and standard errors of each set of DVs (ie, a parametric sweep) when there are multiple set of DVs.

This paper is organized as follows: We summarize related work in Section 2. We describe our system architecture in Section 3. We present our experimental results in Section 4, and we conclude with future research directions in Section 5.

2. Related works

In this section, we briefly discuss recent developments of grid computing (Section 2.1) and summarize works that specifically address the issues of achieving parallelism for Excel (Section 2.2).

2.1. Grid computing

One of the early approaches in achieving more computing power is to use the network of work-stations as a platform for parallel computation. The middleware such as Parallel Virtual Machine (PVM. Geist *et al*, 1994) can be used to link together a number of workstations on the network. With an Application Programming Interface (API) provided, a parallel program can be developed to utilize computing power of workstations. This approach was subsequently standardized by the MPI Forum, which issues a standard-based *m*essage *p*assing *i*nterface called MPI (Message Passing Interface Forum, 2008). Currently, there are many implementations of the MPI standard: Open-MPI (Graham *et al*, 2005), MPICH (The CH comes from Chameleon which is the portability layer. Gropp *et al*, 1999), and LAM (Local Area Multicomputer. Squyres and Lumsdaine, 2003).

Besides workstations, it is also possible to link together a number of PC systems, each of which may run on different operating systems—Unix, Linux, or Windows—via high-speed networks to form a powerful parallel computing platform. This alternative is increasingly appealing as PC systems become more powerful, and commodity network (eg, Fast Ethernet and Gigabit Ethernet) is financially viable. The NASA's Beowulf Project (Sterling, 2001) is one of the major pioneers of this idea. It successfully demonstrates that a supercomputing-class machine can be built cost effectively from Linux PC clusters and available parallel runtime systems such as PVM and MPI. As a result, the Beowulf cluster is a dominant platform in the Top 500 list of the most powerful computing systems (Top500.org).

High Throughput Computing is another approach in capitalizing computing power by using a batch job scheduler to execute a large number of independent jobs on a high-performance computing system. Many schedulers are designed to efficiently manage computing resources on dedicated (ie, reserved for specific needs) and non-dedicated networks of workstations and PC clusters; for example, Condor (Litzkow *et al*, 1988), SGE (Gentzsch, 2001), Torque (Cluster Resources Inc.), and LSF (Platform Computing). Fault handling, check points, restarts, and job accounting are among the features generally supported. These schedulers can accommodate a large number of jobs and large batch job processing, common in scientific and engineering computation. However, most of them lack the necessary support on integration to modern softwares on the client sites; thus, users may experience difficulty in accessing available computing power.

As the speed of the Internet becomes higher, especially for the wide area network in the research and education community such as Internet2 (Stone, 2004), it becomes possible to link geographically dispersed computing resources over the Internet and to use them in a coordinated manner for computation and storage. Grid computing technology (Foster *et al*, 2001) is based on this idea. The Globus Toolkit (Foster and Kesselman, 1997) is a set of open-source software tools that can be used to connect multiple large-scale computing systems, clusters, and supercomputers together, over the wide area network. Many projects at national and international levels use Globus or the modified versions of the *Globus Toolkit* to build large computational and data grids that can execute at a Teraflops or Petaflops level; for example, TeraGrid in the United States and Enabling Grid for E-ScienceE in the EU. These server grids are designed to provide massive amount of computational power to many users who run sizable science and engineering applications. As a result, these grids are generally large, complex, and required a team of system administrators to ensure the grid's high availability for users.

Conceptually similar to the grid computing is the Internet-based distributed computing technology based on desktop computers, for example, SETI@Home, FightAIDS@home, Folding@home, and World Community Grid by IBM. These projects usually focus on solving one specific large-scale problem such as discovering a cure for AIDS or searching for signs of extraterrestrial intelligence. The computing infrastructure is based on harvesting idle cycles of a massive number of PCs over the Internet. The application task is divided into many small work units with different data sets. Then they are farmed out to individual PCs that are pre-installed with a client software. It is sometimes called *volunteered computing* since each participating PC has to volunteer their idle computing cycles to the project.

Because the volunteered computing utilizes a large number of desktop PCs that are linked together in a grid-like manner, it is also one type of desktop grids. Many middleware technologies have been developed for this Internet-based distributed computing; for example, GridMP (Univa UD), Xgrid (Apple), WinGrid (Mustafee and Taylor, 2009), BOINC (Anderson and Fedak, 2006), and Nimrod (Abramson *et al*, 2000). These tools usually consist of a server system and a large number of client machines. The role of a server is to manage the overall system, to break down a large data set and computation into multiple small work units, and to handle some requests and faults in the system. The client machines receive the data sets and applications from the server and perform the computation for users. Most of the clients only harvest idle time of desktop PCs by waiting quietly in the system and perform the computation when machines are idle for some period of time.

2.2. Spreadsheets and distributed computation

In order to use the desktop grids, users need to develop their own programs using the API provided by the product and then build server and client components for the application. The application development may be challenging for some users, and usability of the application may not be very high. Although many projects integrate Excel as a front end to this class of middleware (eg, *ActiveSheets* by Abramson *et al*, 2001), users still need to develop a program that will be executed on client machines.

ActiveSheets is an application that allows parallel evaluation of spreadsheets. It requires custom functions for parallel calculations that are done on backend computers. When the computation is finished, the results are displayed on spreadsheet. In Abramson *et al* (2001), the backend platform is managed by EnFuzion (www.axceleon.com) on a high performance computing system such as computer clusters or grids. On the other hand, Abramson *et al* (2004) use NetSolve (Agrawal *et al*, 2003) as a grid middleware.

Our S3 system also belongs to this class of computing platform, but a different approach is taken. After the infrastructure is installed, users do not need to develop any application programs; all of the required calculations are defined by users on computation sheets. S3 uses the information that a user has provided on the template to split the computation sheet into small work units and send them to client machines for computation. This helps lower the time to results since no programming is required. Therefore, S3 can provide a simple integrated solution from middleware to spreadsheet simulation; it allows an easy setup and execution of Excel-based spreadsheet calculations for users.

In Mustafee and Taylor (2009), *WinGrid* is used for parallelising Excel-based calculations over a network of computational resources. In one of the case studies, WinGrid enables *Witness*—a commercial simulation package—to perform simulation replications in parallel on enterprise grid (linking resources within the same organization, as opposed to a public grid). In their work, users do not build simulation models on Witness directly but specifying model parameters through Excel. Simulation results are also displayed on Excel.

Nadiminti *et al* (2004) introduce *ExcelGrid*, an open-source .NET plug-in that uses Excel as a front-end to a grid and performs user-defined calculations on it. Microsoft also offers a tool for creating *Excel Services* for running a parametric sweep (ie, complete enumeration) on an Excel 2007 Windows Compute Cluster Server (CCS) 2003 (Microsoft Corp, 2008). Excel Services is an architecture that allows Excel calculation on servers and enables applications to access Excel files. However, Excel Services and Windows CCS are not specifically designed for stochastic simulation. Thus, without further programming, users have no control over random number streams, and they have to implement their own algorithms for generating random variates.

Platform Symphony (www2.platform.com) is a commercial software designed to operate on enterprise grids. Platform introduces *Adapter*, which enables Excel calculations to be run in parallel. Symphony's Adapter is targeted for financial applications.

Widely used commercial Excel add-ins for stochastic simulation also offer parallel versions: @Risk (www.palisade.com) provides RiskAccelerator™, whereas Crystal Ball no longer supports its Crystall Ball Turbo™.

3. Design and architecture

High performance parallel computing technology is not widely adopted for a number of reasons. First, users have to develop a complex parallel program for their simulation algorithms. Second, most of the parallel program development environments do not work well with modern user's interface. Hence, users have to develop their own easy-to-use user's interface. Third, the integration of a parallel execution environment, a parallel program, and user-interface components can be very challenging.

In order to address these issues, our approaches are as follows:

1. We let users implement their simulation models on Excel. Thus, the need to develop complex application code is eliminated. All users have to do is to describe their simulation model through Excel formulas and macros. We develop a computational template for users to build their simulation models in such a way that they can be easily distributed over a desktop grid. Some examples are provided as a proof of concept.
2. We model parallel computation using *domain decomposition* technique (Quinn, 2003), where a parallel application is decomposed into many instances of applications that execute the same computation code, but on a different data set. Hence, Excel simulation models can be

efficiently distributed onto a desktop grid. In this work, we emphasize on the development of a middleware infrastructure that interfaces with Excel and distributes the Excel sheets that users have developed to multiple desktops on a desktop grid.

In the following sections, we explain the design and implementation in more detail.

3.1. Spreadsheet template

One of the key components in the design is the computational template on the Excel format. The function of the template is to assist users in defining inputs, outputs, and simulation models such that our middleware can extract this information for computation. Our spreadsheet template is designed for four types of problems:

1. *Deterministic models*:
   (a) with multiple set of inputs, for example, estimating stock betas from historical prices;
   (b) with lower and upper bound of discrete-valued inputs, for example, sensitivity analysis for a linear programming problem;
2. *Stochastic simulation models*:
   (a) with single set of DVs, for example, a transportation problem where some cost coefficients are not constant, such as the Applichem problem in Section 4.1;
   (b) with lower and upper bound of discrete-valued DVs, for example, estimating the cost of a (s, S) inventory problem for every inspection level s and the order-up-to level S in the range of possible values.

Because Case 1 is a special case of Case 2, we only explain our design for the simulation template. In S3 system, simulation replications (ie., runs) of the same model yield outputs that are independent and identically distributed (iid); therefore, we achieve parallelism by assigning simulation replications that use a different set of random inputs to each compute node. Once the calculations at all compute nodes are completed, simulation outputs are then aggregated and summarized, numerically via summary statistics, such as sample means and standard errors, or graphically, through Excel's charting tools.

It can be shown that random numbers of any parametric distributions can be transformed from uniform random numbers over the range $(0, 1)$ (see, for example, Banks *et al*, 2010 for proof). These standard uniform random variables are generated from mathematical algorithms, the so-called *random number generators* (RNGs). RNGs produce a very long sequence of pseudo-random numbers, and RNG seeds allow us to specify from which point in this sequence we get our numbers; we obtain the same sequence of numbers if the seeds are identical (see L'Ecuyer, 2006 and Devroye, 2006 for brief summaries on RNGs and random variate generation, respectively).

Excel has a built-in RNG, called `RAND( )`, which is not used in S3 for the following reasons: we do not know how to control sequence of numbers that `RAND( )` produces. In addition, `RAND( )` has some statistical deficiencies: Knusel (2005) and McCullough and Wilson (2005) discuss `RAND( )` issues in Excel 2003. Therefore, we separate $(0, 1)$ uniform random numbers and Excel outputs from other Excel calculations.

In order to allow S3 to understand the template and extract correct information for parallel execution, we fix the names of the following two Worksheets:

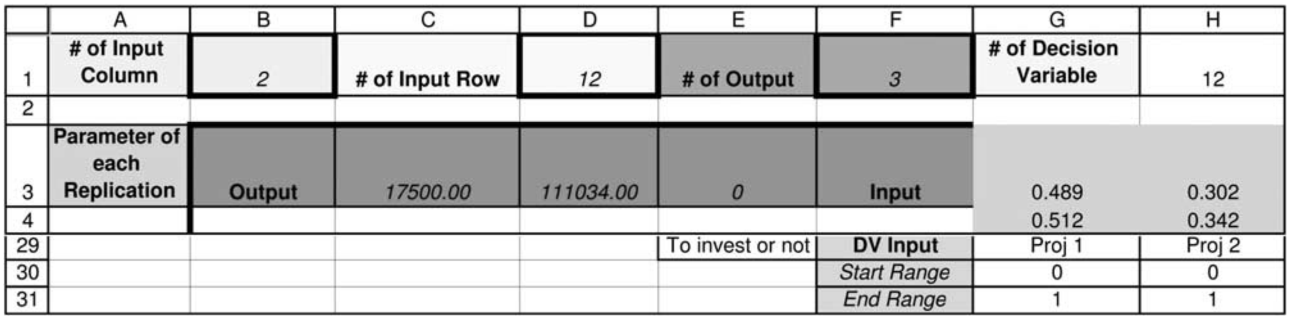
- `Model` performs calculations. If random variates are required, they are generated from our VBA `macro rvg.xla`. These random variate generation functions take uniform$(0, 1)$ random numbers on `Worksheet SimRun` as arguments.
- `SimRun` is a template where users specify details about a simulation model such as number of rows and columns and their locations of uniform$(0, 1)$ random numbers, simulation outputs, and DVs (their respective lower and upper bounds). `SimRun` also holds uniform $(0, 1)$ random numbers that Worksheet `Model` uses. The RNG that our compute nodes return is Mersenne Twister ('`mt19937`') where we adapt to C# from the C code in the GNU Scientific Library (Galassi *et al*, 2006).

Users can also execute simulation runs under multiple sets of DVs by specifying the upper and lower bounds of each DV in Worksheet `SimRun` (see Figure 1). Currently, we only allow integer values with step size of 1. In the project selection example (Section 4.2), our DVs are binary, that is, to invest or not invest on each project. DVs are specified on Worksheet `SimRun`, and they are used in Worksheet `Model` for calculation.

Once the calculation is completed, results are displayed on another Worksheet called `Output` (see Figure 2). For each combination of DVs in the specified range, sample means and standard errors are returned. (Standard error is a measure of how close a sample mean is to the unknown true mean. It is defined as a sample standard deviation divided by square root of the number of replications.)

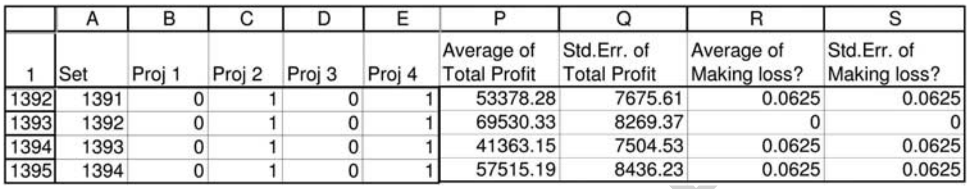
3.2. Implementation of the S3 system

The S3 software architecture is shown in Figure 3. A desktop grid is a basic fabric of the system. On top of the desktop grid, our S3 middleware coordinates between the execution of software components and application on the grid. This software and Excel must be installed on every PC on the grid. The real computation will take place on Excel.



	A	B	C	D	E	F	G	H
1	# of Input Column	2	# of Input Row	12	# of Output	3	# of Decision Variable	12
2								
3	Parameter of each Replication	Output	17500.00	111034.00	0	Input	0.489	0.302
4							0.512	0.342
29					To invest or not	DV Input	Proj 1	Proj 2
30						Start Range	0	0
31						End Range	1	1

Figure 1 Screenshot of Worksheet SimRun.



	A	B	C	D	E	P	Q	R	S
1	Set	Proj 1	Proj 2	Proj 3	Proj 4	Average of Total Profit	Std.Err. of Total Profit	Average of Making loss?	Std.Err. of Making loss?
1392	1391	0	1	0	1	53378.28	7675.61	0.0625	0.0625
1393	1392	0	1	0	1	69530.33	8269.37	0	0
1394	1393	0	1	0	1	41363.15	7504.53	0.0625	0.0625
1395	1394	0	1	0	1	57515.19	8436.23	0.0625	0.0625

Figure 2 Screenshot of Worksheet Output.

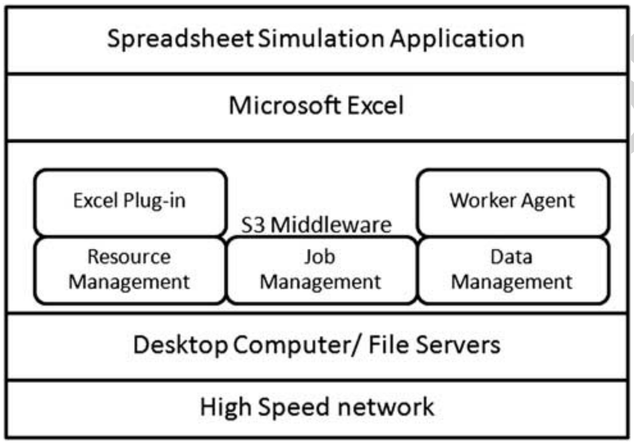


Figure 3 S3 architecture.

The S3 system consists of four components as shown in Figure 4. The role of each group is described below:

1. User's desktop allows users to upload Excel simulation models and download Excel output files when jobs are completed.
2. Manager is responsible for resource management (bookkeeping of workers'status), job management (job submission, job scheduling, and job allocation), and data management (managing data files).
3. Workers or compute nodes are PCs that execute Excel calculations. Currently, we have *dedicated workers* which are always available for the Manager even if there are other jobs running on them. We install a software agent that manages each worker and interacts with the Manager.
4. File Server stores data files that are created during job execution. Users upload Excel files that contain their simulation models onto this File Server from which workers subsequently download. Once simulation is finished, users download Excel files that hold simulation results from the File Server.

We need File Server and Manager in the system because we want to build a system that can perform complex control over computational resources. By not relying on users' computers, the system is more stable and predictable. The existence of File Server eliminates the need for the users' computers to manage complex communication sessions with many workers.

All of the communications among components use .NET Web Services and the FTP protocol. Our numerical experiments show that even without a proprietary high-performance protocol, our proposed system can achieve good performance. The .NET and Web Services standards enable us to use modern programming environment such as Microsoft Visual Studio for development, which becomes faster and more reliable. In addition, our software comes with an installer that makes it easy to install client agents on any Windows-based PCs. Manager and File Server can be merged into one if needed. Therefore, the system can be

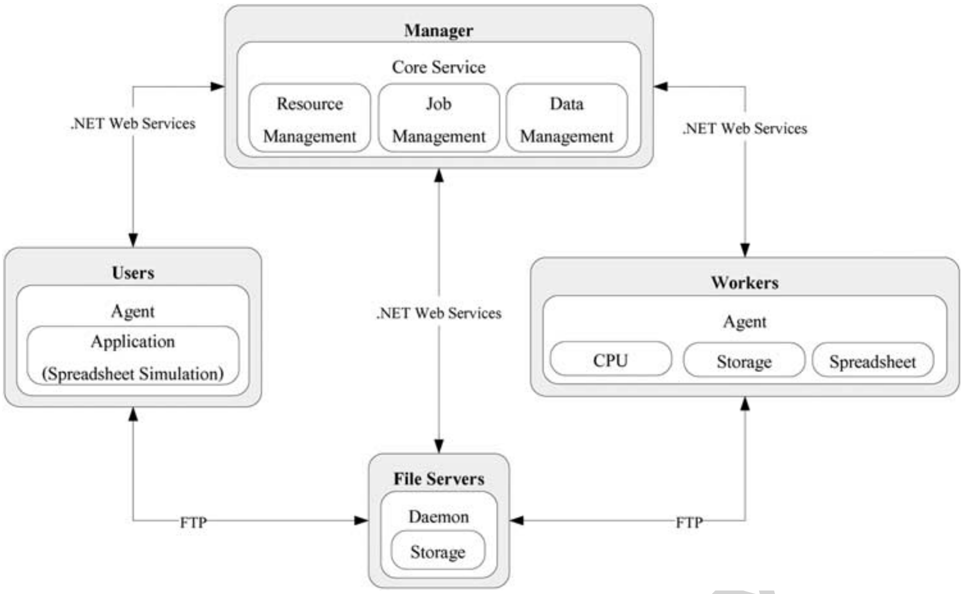


Figure 4 S3 components.

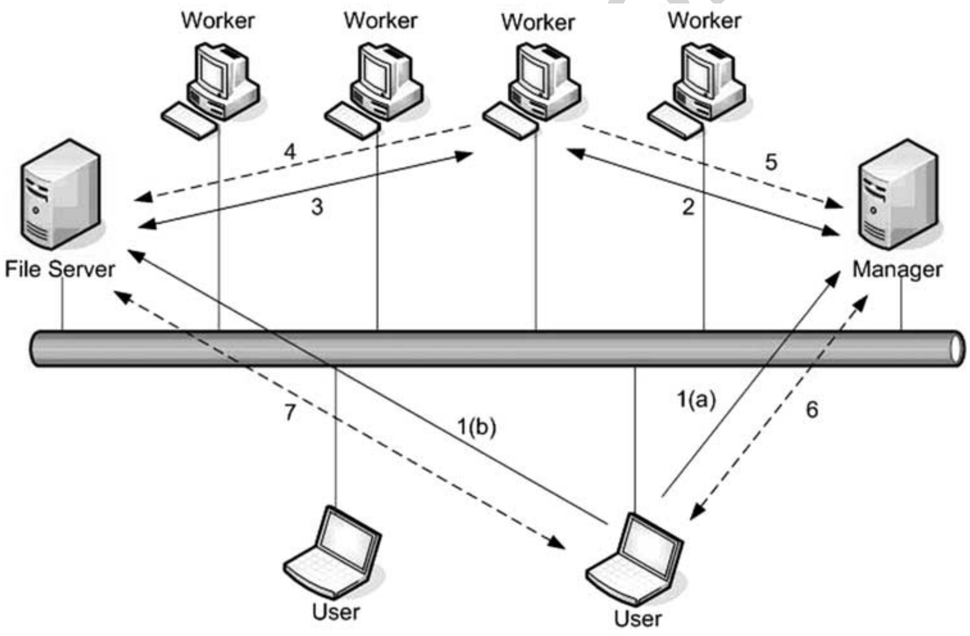
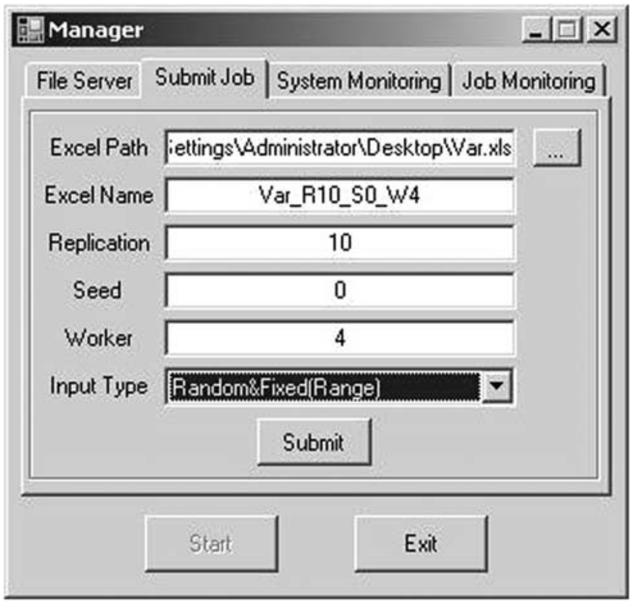


Figure 5 System configuration.

installed quickly without the need to setup a dedicated computing facility for simulation.

During the execution S3, the following steps are executed (the numbers below correspond to ones in Figure 5):

1. A user creates a simulation model in Excel, which has our add-in that connects to Manager through Web Services. Figure 6 shows the dialog box for job submission. Job descriptions (such as a number of replications, a number of workers, and a seed number) are sent to Manager (1(a)) and the user's Excel file is sent to File Server (1(b)).
2. 'Idle' Workers (not currently running Manager's jobs but may be doing other jobs) periodically check with Manager to request jobs. If there are pending jobs, Manager sends them to Workers.
3. A Worker downloads an Excel file according to what Manager has assigned.
4. When a Worker completes his job, Worker uploads his job onto File Server.


Manager

File Server | Submit Job | System Monitoring | Job Monitoring

Excel Path: ;ettings\Administrator\Desktop\Var.xls

Excel Name: Var_R10_S0_W4

Replication: 10

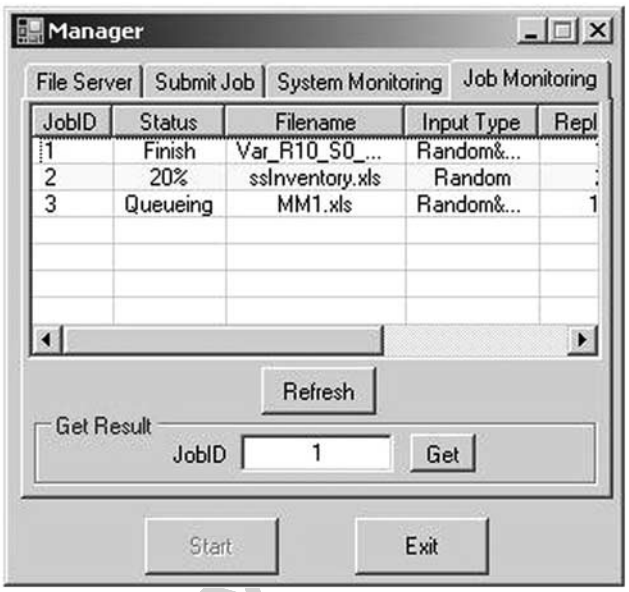
Seed: 0

Worker: 4

Input Type: Random&Fixed(Range)

Submit | Start | Exit

Figure 6 Dialog box when a user requests simulation runs.


Manager

File Server | Submit Job | System Monitoring | Job Monitoring

JobID	Status	Filename	Input Type	Repl
1	Finish	Var_R10_S0_...	Random&...	
2	20%	ssInventory.xls	Random	
3	Queueing	MM1.xls	Random&...	1

Refresh

Get Result: JobID 1 Get

Start | Exit

Figure 7 Dialog box when a user checks job status.

5. Worker updates his status with Manager.
6. A user can check the status of his submitted jobs through Manager (see Figure 7).
7. When the user's job is completed, he downloads his Excel file output from File Server. Figure 2 shows an example of output display.

4. Experimental results

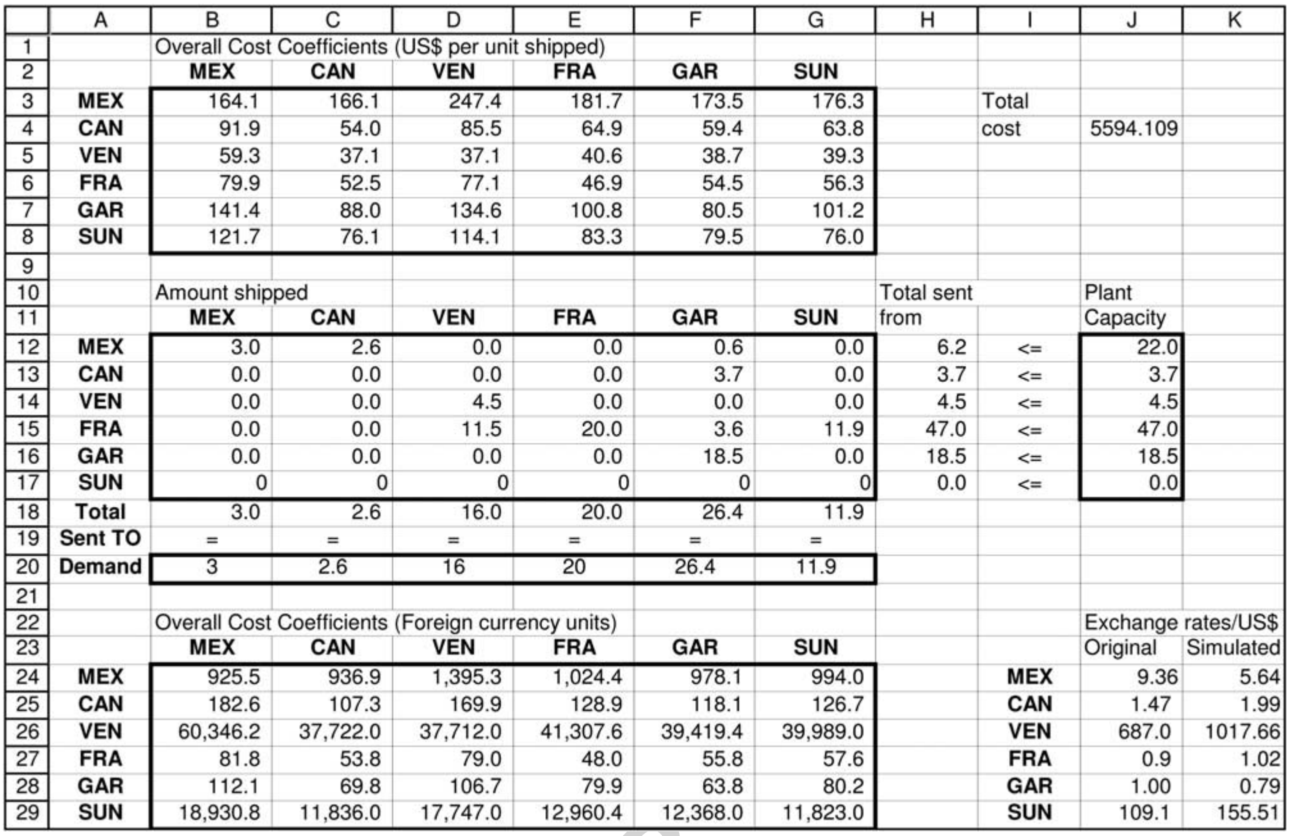
The experiment is done on a system with 11 PCs. The user's system is an Intel Celeron 2.53 GHz system with 512 MB RAM installing Windows XP. Both the Manager and File Server systems are Intel Celeron 2.53 GHz with 512 MB RAM installing Windows Server 2003. The rest of computing nodes are Intel Celeron 2.53 GHz system with 512 MB RAM installing Windows XP. All machines are connected using 100 Mbps Fast Ethernet switch. In this work, all the softwares are developed with Visual Studio 2005 and Visual Studio Tool for Office. The Manager uses the first-come-first-served algorithm for job assignments.

We vary the problem size and the number of workers used. The performance measure is the run time used to complete the simulation runs. Each experiment is repeated three times to get an average run time. From run time results, we compute *speedup*, which is defined as a ratio between sequential run time (run time on one worker) and parallel run time on multiple workers. Speedup shows how much faster the execution is when parallel computing is used. In addition, the system overhead for parallel computing can be assessed through a ratio called *efficiency*. The efficiency of parallel computation is a speedup ratio divided by the number of workers. Efficiency indicates effectiveness in which our desktop grid is utilized to solve the problem. Due to communication overhead, efficiency is between 0 and 1 (100%), where being closer to one is desirable.

We describe problem setups in Sections 4.1–4.2, followed by the result discussion in Section 4.3.

4.1. Applichem problem

This test problem is taken from LeBlanc and Galbreth (2007). Applichem produces a chemical from many plants in different countries—Mexico, Canada, Venezuela, Frankfurt (Germany), Gary (USA), and Japan—for export and domestic markets. The DVs are shipment amounts between each country pairs (`B12:G17` in Figure 8) so that all the demands (`B20:G20`) are satisfied, and that the plant capacities (`J12:J17`) are not exceeded. The objective is to minimize the total cost (`J4`), which depends on the cost coefficients between each country pairs (`B3:G8`). The cost coefficients in turn depend on the foreign exchange rates (`K24:K29`), which are uncertain. The realized exchange rates are assumed to be uniformly distributed over the range [0.5*Original rate, 1.5*Original rate]. In each simulation replication, all the exchange rates are sampled, Excel's Solver is called, and S3 collects total costs. When S3 finishes, it reports the costs of all replications. For experimental purposes, we vary the problem size through the number of simulation replications for each set of DVs (100, 1000, and 10 000) using different numbers of workers (1, 2, 4, and 8). Figures 9 and 10, and Table 1 show the run time, speedup, and the efficiency results, respectively.



	A	B	C	D	E	F	G	H	I	J	K
1		Overall Cost Coefficients (US$ per unit shipped)									
2		**MEX**	**CAN**	**VEN**	**FRA**	**GAR**	**SUN**				
3	**MEX**	164.1	166.1	247.4	181.7	173.5	176.3		Total		
4	**CAN**	91.9	54.0	85.5	64.9	59.4	63.8		cost	5594.109	
5	**VEN**	59.3	37.1	37.1	40.6	38.7	39.3				
6	**FRA**	79.9	52.5	77.1	46.9	54.5	56.3				
7	**GAR**	141.4	88.0	134.6	100.8	80.5	101.2				
8	**SUN**	121.7	76.1	114.1	83.3	79.5	76.0				
9											
10		Amount shipped						Total sent		Plant	
11		**MEX**	**CAN**	**VEN**	**FRA**	**GAR**	**SUN**	from		Capacity	
12	**MEX**	3.0	2.6	0.0	0.0	0.6	0.0	6.2	<=	22.0	
13	**CAN**	0.0	0.0	0.0	0.0	3.7	0.0	3.7	<=	3.7	
14	**VEN**	0.0	0.0	4.5	0.0	0.0	0.0	4.5	<=	4.5	
15	**FRA**	0.0	0.0	11.5	20.0	3.6	11.9	47.0	<=	47.0	
16	**GAR**	0.0	0.0	0.0	0.0	18.5	0.0	18.5	<=	18.5	
17	**SUN**	0	0	0	0	0	0	0.0	<=	0.0	
18	**Total**	3.0	2.6	16.0	20.0	26.4	11.9				
19	**Sent TO**	=	=	=	=	=	=				
20	**Demand**	3	2.6	16	20	26.4	11.9				
21											
22		Overall Cost Coefficients (Foreign currency units)								Exchange rates/US$	
23		**MEX**	**CAN**	**VEN**	**FRA**	**GAR**	**SUN**			Original	Simulated
24	**MEX**	925.5	936.9	1,395.3	1,024.4	978.1	994.0		**MEX**	9.36	5.64
25	**CAN**	182.6	107.3	169.9	128.9	118.1	126.7		**CAN**	1.47	1.99
26	**VEN**	60,346.2	37,722.0	37,712.0	41,307.6	39,419.4	39,989.0		**VEN**	687.0	1017.66
27	**FRA**	81.8	53.8	79.0	48.0	55.8	57.6		**FRA**	0.9	1.02
28	**GAR**	112.1	69.8	106.7	79.9	63.8	80.2		**GAR**	1.00	0.79
29	**SUN**	18,930.8	11,836.0	17,747.0	12,960.4	12,368.0	11,823.0		**SUN**	109.1	155.51

Figure 8 Applichem LP model.

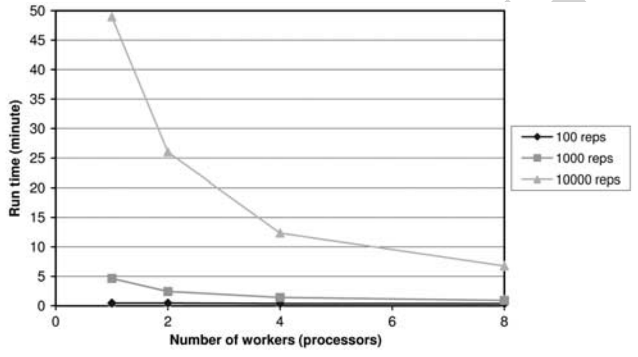


Figure 9 Run time for the Applichem problem when the number of workers and the problem sizes vary.

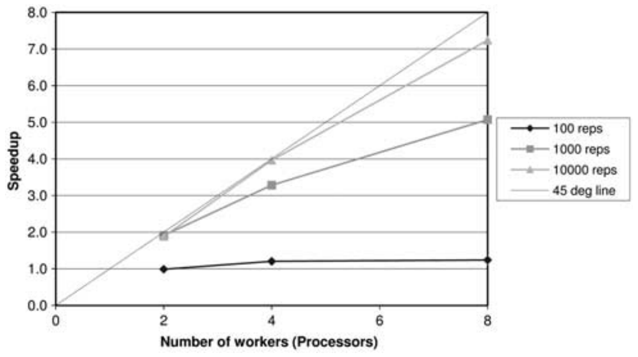


Figure 10 Speedup for the Applichem problem when the number of workers and the problem sizes vary.

4.2. Project selection problem

The company has choices of new projects to invest (Ragsdale, 2004). Each project is undertaken or not undertaken (0–1 DVs; `C6:C17` in Figure 11), and it incurs an initial cost (`D6:D17`). If a project is successful, its revenue is a triangular distributed random variable with parameters as specified in `G6:I17`. The goal is to determine the subset of projects to invest such that the expected total profit (`K18`) is maximized while the investment budget ($80 500) is not exceeded. In each simulation replication, the outcomes that chosen projects are successful or not and its associated revenues are sampled. S3 collects outputs on total profits (revenue—investment cost) and the outcome of making a loss. Because there are 12 projects to consider, we have $2^{12}=4096$ scenarios. We vary the problem size through the number of simulation replications for each scenario (16, 32, 64, and 128) and the number of workers (1, 2, 4, and 8). Figures 12 and 13, and Table 2 show the runtime, speedup, and the efficiency results, respectively.

4.3. Discussion of results

From the numerical experiments, we observe the following characteristics:

- Figures 9 and 12 show that the run time decreases when the number of processing nodes increases, as expected. This is due to the distribution of the processing tasks to multiple computing nodes simultaneously. However, benefits of adding workers increasingly diminishes, for example, the run time decreases sharply when we include the second worker, but not so much when the number of workers increases from four to eight.
- Speedup increases at a faster rate when computing workers are added (Figures 10 and 13). Maximum speedup gained in this experiment is 7.3, which means that the simulation experiment runs seven times faster for the system of only eight computing nodes, that is, 1-h calculation (if done on one PC) can be finished in only 9 min. Thus, for small to moderate problems, what-if analysis or sensitivity analysis is possible.

Table 1 Efficiency for the Applichem problem when the number of workers and the problem sizes vary

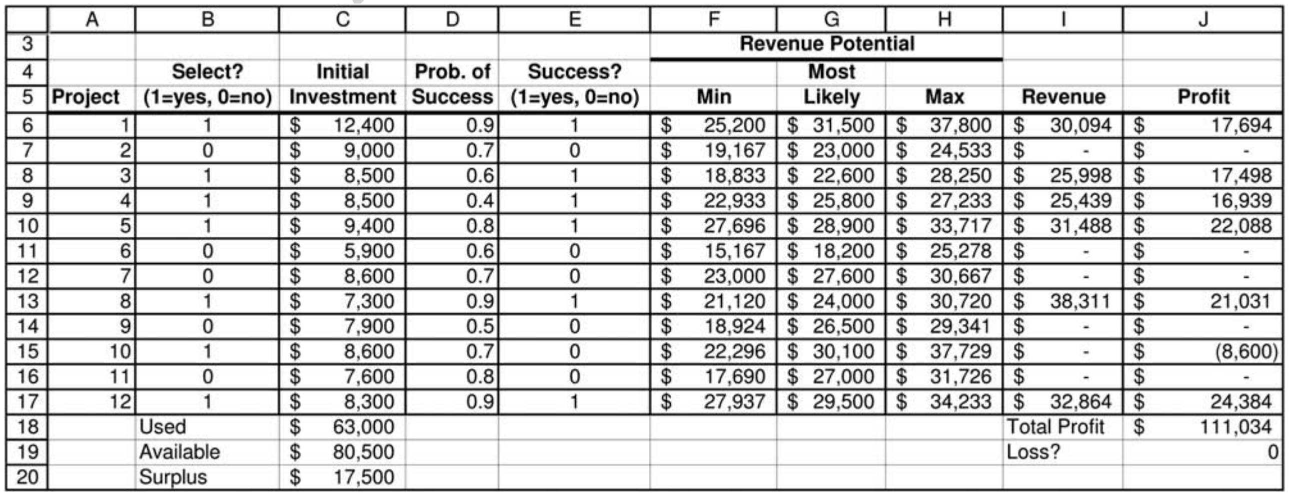
Number of workers	*Number of replications*		
	100	1000	10 000
2	49%	96%	94%
4	30%	82%	99%
8	15%	63%	90%

- Speedup for large problems is higher than for small problems. In Figures 10 and 13, the simulation experiment with the highest number of replications per scenario consistently have the highest speedup for all numbers of workers. When computing workload is large, the fraction of computing overhead (eg, in load balancing, communication, file uploading, and downloading) to the computing workload is lower. Thus, our proposed system will work even better for larger and more complex problems.
- Tables 1 and 2 show the efficiency of our desktop grid. For a given problem size, the efficiency is high for a small number of workers since communication cost is low. As the number of workers increases, the communication overhead increases as well. The efficiency is decreasing accordingly, and at a higher rate. Thus, running a larger simulation model is more efficient because the ratio of computation time to communication time is higher. We can also see that our implementation is very efficient since we can still maintain efficiency of more than 90% with the largest problem sizes of both test problems.

5. Conclusion and future work

We propose an architecture that allows spreadsheet models to utilize a desktop grid to accelerate the execution speed. By employing readily available office PCs, this infrastructure can be built cost effectively. Our approach is novel in that all computation, except for random number generation, is done on Excel. Thus, no complex programming is required.

With S3, we show that the complexity of parallel computing can be mostly hidden from users through well-designed computation templates on Excel. These templates give users flexibility in modelling a simulation problem while enjoying massive computing power. From the experiments, we show that runtime can be reduced sevenfolds with the 8-PC



	A	B	C	D	E	F	G	H	I	J
3						Revenue Potential				
4		Select?	Initial	Prob. of	Success?		Most			
5	Project	(1=yes, 0=no)	Investment	Success	(1=yes, 0=no)	Min	Likely	Max	Revenue	Profit
6	1	1	$ 12,400	0.9	1	$ 25,200	$ 31,500	$ 37,800	$ 30,094	$ 17,694
7	2	0	$ 9,000	0.7	0	$ 19,167	$ 23,000	$ 24,533	$ -	$ -
8	3	1	$ 8,500	0.6	1	$ 18,833	$ 22,600	$ 28,250	$ 25,998	$ 17,498
9	4	1	$ 8,500	0.4	1	$ 22,933	$ 25,800	$ 27,233	$ 25,439	$ 16,939
10	5	1	$ 9,400	0.8	1	$ 27,696	$ 28,900	$ 33,717	$ 31,488	$ 22,088
11	6	0	$ 5,900	0.6	0	$ 15,167	$ 18,200	$ 25,278	$ -	$ -
12	7	0	$ 8,600	0.7	0	$ 23,000	$ 27,600	$ 30,667	$ -	$ -
13	8	1	$ 7,300	0.9	1	$ 21,120	$ 24,000	$ 30,720	$ 38,311	$ 21,031
14	9	0	$ 7,900	0.5	0	$ 18,924	$ 26,500	$ 29,341	$ -	$ -
15	10	1	$ 8,600	0.7	0	$ 22,296	$ 30,100	$ 37,729	$ -	$ (8,600)
16	11	0	$ 7,600	0.8	0	$ 17,690	$ 27,000	$ 31,726	$ -	$ -
17	12	1	$ 8,300	0.9	1	$ 27,937	$ 29,500	$ 34,233	$ 32,864	$ 24,384
18		Used	$ 63,000						Total Profit	$ 111,034
19		Available	$ 80,500						Loss?	0
20		Surplus	$ 17,500							

Figure 11 Project Selection model.

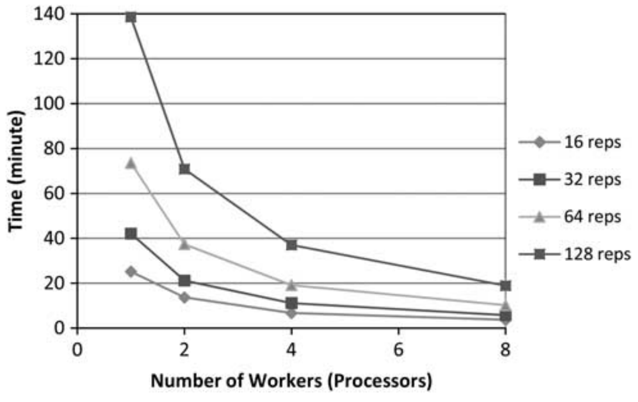


Figure 12 Run time for the project selection problem when the number of workers and the problem sizes vary.

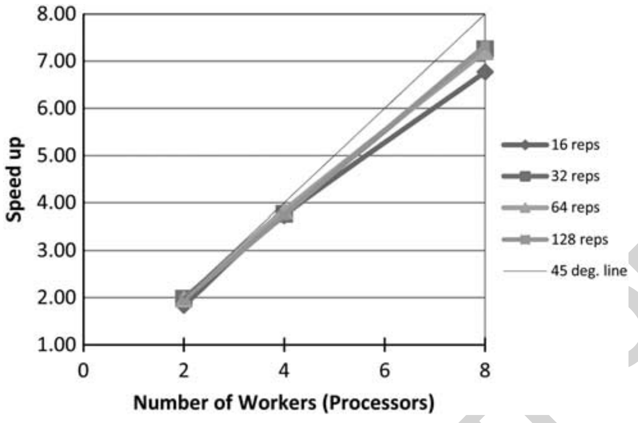


Figure 13 Speedup for the project selection problem when the number of workers and the problem sizes vary.

Table 2 Efficiency for the project selection problem when the number of workers and the problem sizes vary

Number of workers	*Number of replications*			
	16	32	64	128
2	92%	99%	98%	98%
4	93%	94%	96%	93%
8	85%	91%	90%	92%

system. This speedup can make a huge difference in how users analyse problems; with desktop grids, they are able to consider many scenarios simultaneously or even to optimize a sizable model, thus gaining more benefit from their modelling efforts.

Our work can be enhanced in many ways. More templates can be added for broader classes of problems. More transparency can be built so that users are not aware of the existence of Manager by adding automatic job submission into Excel. Load balancing and fault handling can also be added to the S3 system by adopting the 'task pool model'. The execution of a number of spreadsheet instances can be viewed as a pool of tasks being submitted to workers. For the S3 system, faults can arise from many sources: workers, a network, or processing delays that occur when worker nodes or a network experience heavy-load conditions.

Faults can be handled in many ways: the manager may be provided with a simple tracking list of jobs in a queue, jobs being processed, and finished jobs. A timer can be set such that after a pre-specified period of time has elapsed but a job is not finished, it can be resubmitted to a different worker. If a worker or the network is erroneous, the final result will still arrive from this recently released job. If neither the worker nor the network is at fault, but merely slowing down, it is possible that two solutions will be returned to the manager. In this case, the first arriving solution will be saved, and the second one will be discarded. Hence, the overall computation will run to completion regardless of the network's or worker's faults.

Acknowledgements—Portions of this work previously appeared in Pichitlamken *et al* (2008). This research is sponsored by the Thailand Research Fund Grant Number MRG4980121, and it is a part of the Thai National Grid Project supported by Ministry of Information and Communication Technology, the Royal Thai Government. The testing facility is supported by Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Thailand. The authors thank referees for careful readings and helpful comments.

References

Abramson D *et al* (2004). Simplified grid computing through spreadsheets and NetSolve. *Proceedings of the Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region* (*HPCAsia'04*), IEEE Computer Society, Washington, DC, USA: IEEE Computer Society Press, pp 19–24.

Abramson D, Giddy J and Kotler L (2000). High performance parametric modeling with Nimrod/G: Killer application for the global grid? *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, IEEE Computer Society; Washington, DC, USA, p 520.

Abramson D, Roe P, Kotler L and Mather D (2001). Active sheets: Super-computing with spreadsheets. In: *Proceedings of the High Performance Computing Symposium* (*HPC'01*), *Advanced Simulation Technologies Conference*, Society for Modeling and Simulation (SCS) Press; San Diego, California; IEEE Computer Society Press, pp 110–115.

Adams J, Flatto J and Gardner L (2005). Combining hands-on, spreadsheet and discrete event simulation to teach supply chain management. In: Kuhl M, Steiger N, Armstrong F and Joines JA (eds). *Proceedings of the 2005 Winter Simulation Conference*, Institute of Electrical and Electronics Engineers, Inc; Piscataway, New Jersey, pp 2329–2337.

Agrawal S, Dongarra J, Seymour K and Vadhiyar S (2003). NetSolve: Past, present, and future; a look at a grid enabled server. In: Berman F, Fox G and Hey T (eds). *Grid Computing: Making the Global Infrastructure a Reality*. Chichestor, UK: John Wiley & Sons.

Anderson DP (2004). BOINC: A system for public-resource computing and storage. *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society: Washington, DC, USA.

Anderson DP and Fedak G (2006). The computational and storage potential of volunteer computing. *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, IEEE Computer Society: Washington, DC, USA, pp 73–80.

Andradóttir S (2006). Chapter 20: An overview of simulation optimization via random search. In: Henderson SG and Nelson BL (eds). *Handbooks in Operations Research and Management Science, Volume 13: Simulation*. Springer-Verlag: The Netherlands, pp 617–632.

Banks J, Carson II JS, Nelson BL and Nicol DM (2010). *Discrete-event System Simulation* 5nd edn, Pearson: New Jersey, USA.

Boer C, de Bruin A and Varbraeck A (2009). A survey of distributed simulation in industry. *J Simulation* **3**: 3–16.

Cunha CB and Mutarelli F (2007). A spreadsheet-based optimization model for the integrated problem of producing and distributing a major weekly newsmagazine. *Eur J Opl Res* **176**: 925–940.

de Mesquita MA and Hernandez AE (2005). Discrete-event simulation of queues with spreadsheet: A teaching case. In: Perrone L, Wieland F, Liu J, Lawson B, Nicol D and Fujimoto R (eds). *Proceedings of the 2005 Winter Simulation Conference*, Institute of Electrical and Electronics Engineers, Inc; Piscataway, New Jersey, pp 2277–2283.

Devroye L (2006). Nonuniform random number generation. In: Henderson SG and Nelson BL (eds). *Handbooks in Operations Research and Management Science, Volume 13: Simulation* Chapter 4. pp 83–122.

Ecklund P (2007). Notes on excel calculations. Available via, http://faculty.fuqua.duke.edu/_pecklund/ExcelReview/ExcelFormulasReview.pdf, accessed 11 March 2008.

Foster I and Kesselman C (1997). Globus: A metacomputing infrastructure toolkit. *Int J High Perform C* **11**(2): 115–128.

Foster I, Kesselman C and Tuecke S (2001). The anatomy of the grid: Enabling scalable virtual organizations. *Int J High Perform C* **15**(3): 200–222.

Galassi M *et al* (2006). *GNU Scientific Library Reference Manual* 2nd edn, UK: Network Theory Ltd.

Geist A *et al* (1994). *PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing*. Scientific and Engineering Series. Cambridge, MA, USA: MIT Press.

Gentzsch W (2001). Sun grid engine: Towards creating a compute power grid. *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, IEEE Computer Society; Washington, DC, USA, pp 35–36.

Graham RL, Woodall TS and Squyres JM (2005). *Open MPI: A Flexible High Performance MPI*. Springer Berlin/Heidelberg: Poznan, Poland.

Gropp W, Lusk E and Thakur R (1999). *Using MPI-2: Advanced Features of the Message-passing Interface*. MIT Press: Cambridge, MA, USA.

Knusel L (2005). On the accuracy of statistical distributions in Microsoft Excel 2003. *Comput Stat Data An* **48**(3): 445–449.

LeBlanc LJ and Galbreth MR (2007). Implementing large-scale optimization models in excel using VBA. *Interfaces* **37**(4): 370–382.

L'Ecuyer P (2006). Uniform random number generation. In: Henderson SG and Nelson BL (eds). *Handbooks in Operations Research and Management Science, Volume 13: Simulation* Chapter 3. Springer-Verlag: The Netherlands, pp 55–82.

Litzkow MJ, Livny M and Mutka MW (1988). Condor-a hunter of idle workstations. *Proceedings of the 8th International Conference on Distributed Computing Systems*, IEEE Computer Society Press: Los Alamitos, CA, USA, pp 104–111.

Luce K, Trepanier L, Ciochetto F and Goldman L (2005). Simulation and optimization as effective DFSS tools. In: Kuhl M, Steiger N, Armstrong F and Joines JA (eds), 1993–1999, *Proceedings of the 2005 Winter Simulation Conference*, Institute of Electrical and Electronics Engineers, Inc; Piscataway, New Jersey.

McCullough B and Wilson B (2005). On the accuracy of statistical procedures in Microsoft Excel 2003. *Comput Stat Data An* **49**(4): 1244–1252.

Message Passing Interface Forum (2008). *MPI: A Message-passing Interface Standard-version 2.1*. High-Performance Computing Center: Stuttgart.

Microsoft Corp (2008). Microsoft excel running on Microsoft compute cluster, http://msdn.microsoft.com/en-us/library/bb463068.aspx, accessed 7 July 2008.

Mustafee N and Taylor SJE (2009). Speeding up simulation applications using WinGrid. *Concurr Comp: Pract E* **21**(11): 1504–1523.

Myerson RB (2004). *Probability Models for Economic Decisions*. USA: Duxbury Press.

Nadiminti K *et al* (2004). ExcelGrid: A. NET plug-in for outsourcing Excel spreadsheet workload to enterprise and global grids. *Proceedings of the 12th International Conference on Advanced Computing and Communication (ADCOM 2004)* IEEE Computer Society Press: Ahmedabad, India, http://www.gridbus.org/papers/ExcelGrid.pdf, accessed 11 March 2008.

Paisittanand S and Olson DL (2006). A simulation study of IT outsourcing in the credit card business. *Eur J Opl Res* **175**(2): 1248–1261.

Pichitlamken J, Kajkamhaeng S and Uthayopas P (2008). High performance spreadsheet simulation on a desktop grid. In: Mason S, Hill R, Moench L, Rose O, Jefierson T and Fowler J (eds). *Proceedings of the 2008 Winter Simulation Conference*, Institute of Electrical and Electronics Engineers, Inc; Piscataway, New Jersey, pp 663–670.

Quinn M (2003). *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill: New York NY, USA.

Ragsdale CT (2004). *Spreadsheet Modeling & Decision Analysis* 4th edn. South-Western (Thomson Learning): Mason, OH.

Sakalli US and Birgoren B (2009). A spreadsheet-based decision support tool for blending problems in brass casting industry. *Comput Ind Eng* **56**: 724–735.

Seila AF (2006). Spreadsheet simulation. In: Perrone L, Wieland FP, Liu J, Lawson BG, Nicol DM and Fujimoto RM (eds). *Proceedings of the 2006 Winter Simulation Conference*, Institute of Electrical and Electronics Engineers, Inc.; Piscataway, New Jersey, pp 11–18.

Seila AF, Ceric V and Tadikamalla P (2003). *Applied Simulation Modeling*. Thomson Learning: USA.

Shariff AM, , Rusli R, Leong CT, Radhakrishnan V and Buang A (2006). Inherent safety tool for explosion consequences study. *J Loss Prevent Proc* **19**(5): 409–418.

Squyres JM and Lumsdaine A (2003). September/October. A component architecture for LAM/MPI. In: Dongarra J, Laforenza and Orlando S, (eds). *Proceedings of 10th European PVM/MPI Users' Group Meeting*. Number 2840 in Lecture Notes in Computer Science. Germany: Springer-Verlag, pp 379–387.

Ståhl I (2005). Using discrete event simulation in the teaching of decision analysis. In: Kuhl M, Steiger N, Armstrong F and Joines JA (eds). *Proceedings of the 2005 Winter Simulation Conference* Institute of Electrical and Electronics Engineers, Inc.; Piscataway, NJ, pp 2280–2289.

Sterling T (2001). *Beowulf Cluster Computing with Linux*. MIT Press: Cambridge, MA, USA.

Stevenson WJ and Ozgur C (2007). *Introduction to Management Science with Spreadsheets*. McGraw-Hill/Irwin: New York.

Stone A (2004). Internet2's breakthroughs for academic research. *IEEE Distributed Systems On-line* **5**(1): 3.

Taylor S and Robinson S (2006). So where to next? A survey of the future for discrete-event simulation. *J Simulation* **1**: 1–6.

Received 18 August 2009;
accepted 3 September 2010 after one revision