รายงานวิจัยฉบับสมบูรณ์

โครงการ การออกแบบและพัฒนาซอฟท์แวร์สำหรับการจำลองสถานการณ์แบบสุ่มด้วยสเปรดชีท

โดย ผศ. ดร. จุฑา พิชิตลำเค็ญ และคณะ

ตุลาคม 2551

สัญญาเลขที่ MRG4980121

รายงานวิจัยฉบับสมบูรณ์

โครงการ การออกแบบและพัฒนาซอฟท์แวร์สำหรับการจำลองสถานการณ์แบบสุ่มด้วยสเปรดชีท

คณะผู้วิจัย	**สังกัด**
1. ผศ.ดร. จุฑา พิชิตลำเค็ญ	ภาควิชาวิศวกรรมอุตสาหการ
	คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์
2. นายศุภสิทธิ์ กาจกำแหง	ศูนย์วิจัยระบบคอมพิวเตอร์สมรรถนะสูงและเครือข่ายคอมพิวเตอร์
	คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์
3. ผศ. ดร. ภุชงค์ อุทโยภาศ	ศูนย์วิจัยระบบคอมพิวเตอร์สมรรถนะสูงและเครือข่ายคอมพิวเตอร์
	คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์

สนับสนุนโดยสำนักงานคณะกรรมการการอุดมศึกษา และสำนักงานกองทุนสนับสนุนการวิจัย

(ความเห็นในรายงานนี้เป็นของผู้วิจัย สกอ. และ สกว. ไม่จำเป็นต้องเห็นด้วยเสมอไป

เอกสารแนบหมายเลข 2

รูปแบบ Abstract (บทคัดย่อ)

Project Code (รหัสโครงการ) : MRG4980121

Project Title: การออกแบบและพัฒนาซอฟท์แวร์สำหรับการจำลองสถานการณ์แบบสุ่มด้วยสเปรดชีท

Design and Development of **S**tochastic **S**imulation Tool for **S**preadsheet (S3)

Investigators:

1. ผศ.ดร. จุฑา พิชิตลำเค็ญ	ภาควิชาวิศวกรรมอุตสาหการ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์
2. นายศุภสิทธิ์ กาจกำแหง	ศูนย์วิจัยระบบคอมพิวเตอร์สมรรถนะสูงและเครือข่ายคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์
3. ผศ. ดร. ภุชงค์ อุทโยภาศ	ศูนย์วิจัยระบบคอมพิวเตอร์สมรรถนะสูงและเครือข่ายคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์

E-mail Address: juta.p@ku.ac.th

Project Period (ระยะเวลาโครงการ): 1 กรกฎาคม 2549 ถึง 30 มิถุนายน 2551

Abstract

We present a proof-of-concept prototype for high performance spreadsheet simulation called S3. Our goal is to provide a user-friendly, yet computationally powerful simulation environment for end users. Our approach is to add power of parallel computing on Windows-based desktop grid into popular Excel models. We show that, by using standard Web Services and Service-Oriented Architecture (SOA), one can build a fast and efficient system on a desktop grid for simulation. The complexity of parallelism can be hidden from users through a well-defined computation template. This work also demonstrates that a massive computing power can be harvested by linking off-the-shelf office PCs into a desktop grid for simulation. The experimental results show that the prototype system is highly scalable. In the best case, the execution time can be reduced 13.6 times using 16 desktop PCs; the simulation time is dramatically reduced from 200 minutes to 14 minutes.

Keywords: Spreadsheet simulation, simulation modeling, grid computing, parallel computing, desktop grid.

บทคัดย่อ

ผู้วิจัยนำเสนอซอฟท์แวร์ต้นแบบเพื่อการจำลองสถานการณ์บนสเปรดชีท ที่มีสมรรถนะสูง โดยใช้ชื่อว่า S3 งานวิจัยนี้มีจุดมุ่งหมายเพื่อพัฒนาระบบที่คำนวณได้รวดเร็ว และง่ายต่อการใช้งานสำหรับผู้ใช้งานจริง ที่อาจไม่ใช่โปรแกรมเมอร์ การคำนวณแบบขนานของระบบกริด ที่สร้างจากเดสก์ท็อบพีซี สามารถทำให้ตัวแบบเอ็กเซลถูกประมวลผลได้เร็วขึ้น เอ็กเซลมีข้อดีตรงที่เป็นซอฟท์แวร์ที่รู้จักกันดี และเป็นเครื่องมือสำหรับการวิเคราะห์ ด้วยงานวิจัยนี้ ผู้วิจัยได้แสดงให้เห็นว่า มาตรฐานของเว็บเซอร์วิส และ Service-oriented architecture ช่วยให้สามารถสร้างระบบการคำนวณเพื่อใช้ในการจำลองสถานการณ์ ที่คำนวณได้รวดเร็ว และมีประสิทธิภาพ บนเดสก์ท็อบกริด ความซับซ้อนของการคำนวณแบบขนานสามารถถูกซ่อนไม่ให้ผู้ใช้รับรู้ โดยการใช้เทมเพลทสำหรับการคำนวณที่ถูกออกแบบมาอย่างเหมาะสม นอกจากนี้ งานวิจัยนี้ยังแสดงให้เห็นว่า เมื่อระบบการคำนวณที่มีสมรรถนะสูงเพื่อการจำลองสถานการณ์ สามารถถูกสร้างจากเดสก์ท็อบพีซีที่ใช้ในออฟฟิศทั่วไป ผลการทดลองแสดงให้เห็นว่า ระบบต้นแบบสามารถรองรับปัญหาที่มีขนาดใหญ่ขึ้นได้ ในกรณีที่ดีที่สุด เวลาที่ใช้ในการคำนวณถูกลดลง 13.6 เท่า เมื่อใช้ระบบที่ประกอบด้วยเดสก์ท็อบพีซี 16 ตัว เวลาที่ใช้ลดลงจาก 200 นาที เหลือเพียง 14 นาที

(คำหลัก) การจำลองสถานการณ์บนสเปรดชีท การจำลองสถานการณ์ การคำนวณบนกริด การคำนวณแบบขนาน เดสก์ทอบกริด

เอกสารแนบหมายเลข 3

Output จากโครงการวิจัยที่ได้รับทุนจาก สกว.

1. **Pichitlamken, J.**, S. Kajkamhaeng, and P. Uthayopas. High Performance Spreadsheet Simulation on a Desktop GRID. In *Proceedings of the 2008 Winter Simulation Conference*, ed. S. J. Mason, R. R. Hill, L. Moench, and O. Rose. To appear.
2. **Pichitlamken, J.**, P. Uthayopas, S. Kajkamhaeng, and N. Tippayawannakorn. Service-Oriented Architecture on a Windows Cluster for Spreadsheet Simulation. In *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 1751--1761, Singapore.

ภาคผนวก

Service-Oriented Architecture on a Windows Cluster for Spreadsheet Simulation

Juta Pichitlamken[1], Putchong Uthayopas[2], Supasit Kajkamhaeng[2], Noocharin Tippayawannakorn[1]
[1]Department of Industrial Engineering, Kasetsart University, Bangkok, Thailand
[2]High Performance Computing and Networking Center, Kasetsart University, Bangkok, Thailand

Abstract - **We present a proof-of-concept prototype for applying service-oriented architecture (SOA) on a Windows cluster to spreadsheet simulation. A scalable architecture based on Web Services is proposed. Our goal is to provide a user-friendly, yet computationally powerful simulation environment for end users. The experimental results show that the prototype system functions in a highly scalable way.**

Keywords - **service-oriented architecture, Windows cluster, spreadsheet simulation**

I. INTRODUCTION

Simulation models are increasingly being used as an analytical tool because advances in computers, in terms of speed and memory, enables analysts to model complex systems, e.g., military, logistic and manufacturing applications. Nelson [1] observes that "the availability of more computing horsepower has whetted the appetite of modelers to solve more complex problems, more often." Generally, an analyst does not want to just model the system of interest, but her ultimate goal is likely to be finding the best decision (or input variables) for the system of interest. This is often called *optimization-via simulation* problems (see [2], [3] and [4] for recent reviews). If simulation runs for a single scenario takes a long time, analyzing it under multiple ones (i.e., doing a what-if analysis) may be prohibitive in practice.

To get more computing power, one can run a simulation model on a high-end server. Another high-performance but less expensive solution is clustering. A cluster is a group of computers connected via a high-speed network. Computers inside a cluster share resources such as storage, data, software and computing power. When an application (e.g., a simulation model) is running on a cluster, it can harness the shared computing resources that act as a unified pool. A cluster is appealing because it can be built from commodity PCs which are much cheaper than a high-end server.

In this paper, we build a spreadsheet simulation system that is easy to use and computationally fast by utilizing power of a Windows cluster. Microsoft Windows is chosen because it is robust and easy to configure. We have used a similar architecture to develop a real world application for the Thai Bond Market Association (www.thaibma.or.th). In that work, the Value-at-Risk (VaR) is calculated for a portfolio of stocks traded in the Stock Exchange of Thailand (see [5] for more details).

We also employ the service-oriented architecture (SOA) in designing our cluster. He [6] defines SOA as "an architectural style whose goal is to achieve loose coupling among interacting software agents." By applying an SOA principle to a distributed simulation system, we can easily reuse the functionality of existing simulation models and combine them together to model more complex problems.

To achieve the cluster environment, we implement our simulation model as *Services* which is deployed on every computer inside the cluster. In our case, a "Service" is running a simulation model. Using Web Services and a multithreading technique, a client-side module can simultaneously invoke multiple services in parallel to speed up the execution of a simulation model. We also add a broker component that helps locate Services and perform simple load balancing to better utilize the system resources.

The simulation model that we consider is the project selection problem which is implemented in Excel®. Given an investment decision—a combination of projects to invest— the expected profits is estimated via simulation. We choose this problem for the following reasons: It is small and computationally straight-forward, but simulation is required to estimate the expected payoff because it does not have a closed-form expression. In addition, this problem is relevant in practice as it is essentially a resource allocation problem.

This paper is organized as follows: Section II provides background knowledge related to this research. We describe our system architecture in Section III. We present our experimental results in Section IV. We conclude with future research works in Section V.

II. BACKGROUND

Currently, most applications are generally designed to run on a single processor (*sequential programming*). To efficiently utilize the computing resources shared in a cluster, applications must be implemented using parallel programming technique, where processing is distributed on multiple processors in a cluster (see [7] and [8]). Usually, the processors can communicate with each other and exchange some data. A major drawback of parallel/ distributed programming is that it generally takes more time and effort to design and implement applications than sequential programming. Therefore, a good programming environment is crucial for the development of parallel programs.

Widely-used tools for building a parallel programming application include Application Programming Interfaces (APIs), such as Message Passing Interface (MPI, [9]; [10]) and Parallel Virtual Machine (PVM, [11]). These tools are designed to be used effectively on large parallel computer or cluster systems. Recently, modern development frameworks (e.g., J2EE, [12] and .NET framework, [13]) are appealing because they provide many tools and APIs that supports real-world application implementations. These tools have been extensively tested in modern enterprise computing environment (see, for example, [14]). Its benefits include a user-friendly development environment and close conformity to industrial standards. However, a programmer has to understand a good deal of parallel computing techniques to use these tools.

When a cluster is built for parallel programming, two types of operating systems (OSs) are usually selected: Linux and Windows. With Linux OS, open-source softwares and commodity hardwares can be used. This type of cluster is also known as Beowulf clusters [15]. Softwares for building Linux clusters include NPACI Rocks and OSCAR [16]. The appeals of Linux clusters are highly configurable architecture, high scalability, and low software licensing costs. However, a Linux skill is required in using a cluster and performing system administration tasks; therefore, Linux clusters are found mostly in technical and academic environments where the required skills are abundant.

Another main alternative is to build a Windows cluster. Microsoft Corp. provides a clustering solution via the Microsoft Windows OS. There are many benefits from the use of Windows cluster technology. Firstly, a typical user is more likely to be familiar with Windows than with Linux. Secondly, building and maintaining a Windows cluster may be easier than a Linux one, with a rich set of graphic user interface-based tools. In addition, the development time for parallel simulation is faster due to a more powerful programming environment and robust standards, such as Web Services. Finally, to reduce the barrier to adoption for users, complexities related to clustering should be hidden from end users. The use of very familiar tools (such as Excel spreadsheets) as a user interface will ensure that users can readily model problems of interests. Thus, a Windows-based development framework enables seamless integration of parallel capability.

Recently, emerging technologies such as Web Services and XML have led to a paradigm shift in designing a distributed software architecture. Service Oriented Architecture (SOA) can be explained with Fig. 1. The application software consists of 3 main components: *Service Provider*, *Service Consumer*, and *Service Broker*. Instead of building numerous software components, SOA emphasizes on decomposing all major functions (in our case, the simulation models) into network accessible Services that can be invoked using Web Services. As a

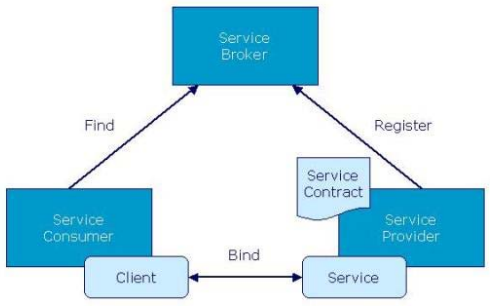


Fig. 1. Service Oriented Architecture concept.

Service comes alive, it will publish its existence to the service broker whose role is to keep track of the existence and status of available Services. See [17] for more details.

Our application is developed as a Service Consumer. When a Service Consumer needs any Services (such as Monte-Carlo simulation), the available Service Providers can be found by sending a finding request to a Service Broker. Then, a Service Consumer will bind itself to the Service Provider and invoke the Service to execute the function on the behalf of a user. The invocation of each Service is done using the Web Service protocol. By applying SOA to simulation, we can potentially populate the network with large number of simulation Services. There are many clear advantages that we can gain from using SOA within simulation contexts. When the need arises, these ready-made Services speed up the simulation tasks by deploying multiple instances of the same Services. Moreover, a complex simulation model can be built by integrating multiple different Services.

III. DESIGN AND ARCHITECTURE

The system architecture used in this work is shown in Fig. 2. The system is composed of three main components. The first part is the *computing Service nodes* which is a set of PCs on which simulation Services are installed. The second component is the *Services broker* or the resource manager that perform two main functions: to locate available machines by keeping track of registration requests; and to perform workload management for the system. The third part is the *Service Consumer* that is developed as an add-in to the Excel spreadsheet. The role of this component is to get a user's request and input data from the spreadsheet model and then invoke the Service on computing Service nodes using Web Service. This Excel add-in is also responsible for creating multiple execution threads and for managing the parallelism to maximize the cluster performance. The execution steps of our architecture can be explained as follows (the numbers below correspond to the ones in Fig. 2):

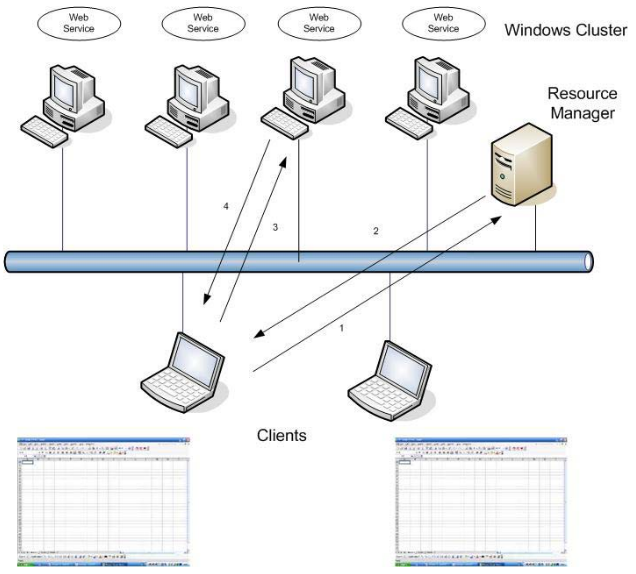


Fig. 2. System configuration.

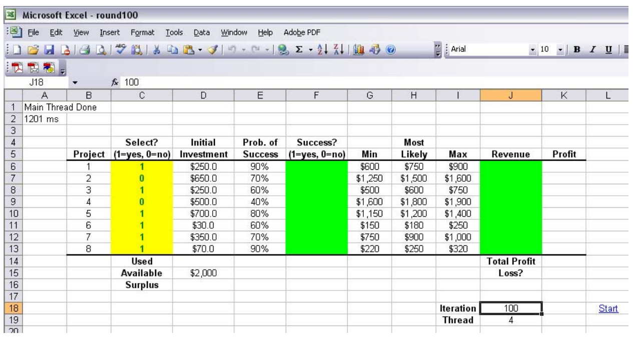
1. A user creates a simulation model in Excel which has our Web Service add-in. When she runs her simulation model, Web Service add-in calls the resource manager to reserve the PC in the Windows cluster.
2. The resource manager tells the client machine which PC is reserved.
3. The Excel add-in in the client machine sends the computation to the reserved PC.
4. Once the computation is completed, the PC in the Windows cluster sends the results back to the client's machine.

IV. Experimental Results

We describe our test problem in Section IVA. The numerical results are shown in Section IVB.

A. Prototype Problem

The problem that we consider is a project selection problem taken from [18]. Given an investment decision—a combination of projects to invest—the expected profits is estimated via simulation. The total investment money is limited to $2 million, and the outcomes of the projects being successful and the project revenues are probabilistic. We model the event that a project is successful or not as a binomial random variable. If the project is successful, the realized revenue is modeled with triangular distribution. The output of each replication is total profit, the amount of investment money used and the surplus investment money. Table I shows numerical input data for this model. Fig. 3 shows the screen shot of the test application.


Microsoft Excel - round100
File Edit View Insert Format Tools Data Window Help Adobe PDF
Arial
10
J18
100

	A	B	C	D	E	F	G	H	I	J	K	L
1	Main Thread Done											
2	1201 ms											
3												
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	$250.0	90%		$600	$750	$900			
7		2	0	$650.0	70%		$1,250	$1,500	$1,600			
8		3	1	$250.0	60%		$500	$600	$750			
9		4	0	$500.0	40%		$1,600	$1,800	$1,900			
10		5	1	$700.0	80%		$1,150	$1,200	$1,400			
11		6	1	$30.0	60%		$150	$180	$250			
12		7	1	$350.0	70%		$750	$900	$1,000			
13		8	1	$70.0	90%		$220	$250	$320			
14			Used							Total Profit		
15			Available	$2,000						Loss?		
16			Surplus									
17												
18									Iteration	100		Start
19									Thread	4		

Fig. 3. Screen shot of the test application.

B. Results and Discussion

The experiment is done on a 5-PC system. The client system is an Intel Pentium III 894MHz system with 256MB RAM installing Windows XP. The spreadsheet, client Web Service add-in and the Service broker executes on this computer. The rest of computing service nodes specifications are: Pentium III 930MHz with 512MB RAM, Pentium IV 3GHz with 512MB RAM, Athlon XP 1GHz with 512MB RAM, and Athlon 64 3000+ with 1GB RAM. All machines are connected together using 100Mbps FastEthernet switch. In this work, all the softwares are developed with Visual Studio 2005 and Visual Studio Tool for Office (VSTO). The resource manager uses the round robin algorithm (see, for example, [19]) to do node assignments.

We control the number of simulation replications and the number of threads used to manage parallelism. The response or the performance measure is the computational time to complete the simulation runs. The test is done at 100, 200, 400, 800, 1600, and 3200 replications. The number of thread used is 1, 2, 4, and 8 threads. The runtime results are shown in Fig. 4.

We see that the run time decreases almost linearly when number of processing nodes increases. This is due to the distribution of the processing task to multiple computing Service nodes simultaneously.

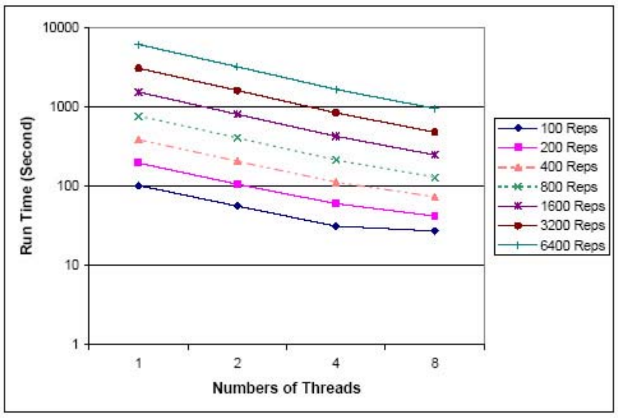


Fig. 4. Runtime on the Windows cluster when the number of simulation replications are varied.

TABLE I
Input data for the project selection example

Project	Initial Investments ($000s)	Probability of success	Revenue potential ($000s)		
			Min	Most likely	Max
1	$250	90%	$600	$750	$900
2	$650	70%	$1250	$1500	$1600
3	$250	60%	$500	$600	$750
4	$500	40%	$1600	$1800	$1900
5	$700	80%	$1150	$1200	$1400
6	$30	60%	$150	$180	$250
7	$350	70%	$750	$900	$1000
8	$70	90%	$220	$250	$320

Fig. 5 shows the speedup of the work. Speedup is defined as a ratio between sequential runtime (i.e., runtime on one thread) and parallel runtime (i.e., runtime on multiple threads). Speedup shows how many times faster the execution is when parallel computing is used. We observe the following results:

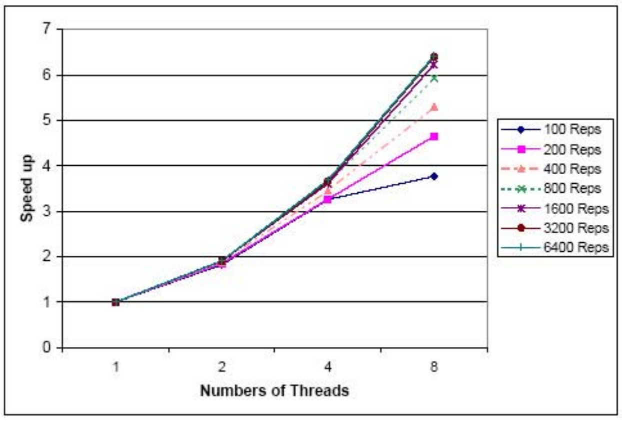


Fig. 5. Speedup on the Windows cluster when the number of simulation replications are varied.

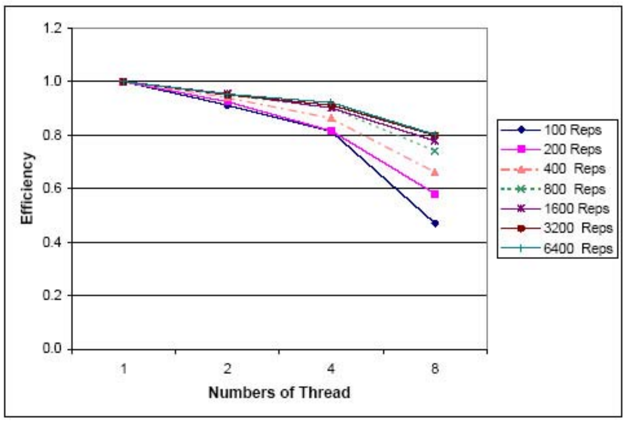


Fig. 6. Efficiency on the Windows cluster when the number of simulation replications are varied.

1. Speed up increases at a faster rate when the number of computing Service nodes are added. Maximum speedup gained in this experiment is 6.4; therefore, the application runs 6-7 times faster for the system of only 8 nodes.
2. Speedup for a larger problem is likely to be higher. The reason is that when more computing workload is available, the fraction of computing overhead (e.g., in load balancing and communication) to the computing workload will be lower. Thus, our proposed system will work even better for large and complex problems.

A fraction of runtime used as system overhead can be assessed through a ratio called *efficiency*. The efficiency of parallel computation is speedup divided by number of computing elements. Fig. 6 shows the efficiency of our cluster.

We can see that for small number of computing Service nodes, the efficiency is high since the communication from application to Services is low. As number of nodes increases, the communication overhead increases as well. Thus, the efficiency is decreasing accordingly, at a higher rate. However, running a larger simulation model may result in a better efficiency because the ratio of computation time to communication time is higher.

V. FUTURE WORKS

We present a prototype of a computing system for spreadsheet simulation. This system is able to utilize a vast amount of computing resources available from a Windows cluster or a farm of desktop PCs. We use Service Oriented Architecture as a basis for the design and capitalize on robust commercial standard technology, such as XML and Web Services. The goal is to provide a user-friendly, yet computationally powerful simulation environment for end users.

The experimental result has shown that our prototype system functions in a highly scalable way. Currently, its major limitation is that the system can only simulate models whose entire simulation replication is on one computing node, i.e., there is no interactions or communications among nodes. The speedup occurs because multiple replications can be done simultaneously.

In the future, we plan to explore the issue of scalability. This can be achieved by using a more efficient load balancing algorithm at the Service broker. As a system scales up, it has a potential to reach out beyond a single physical location. Thus, the issue of reliability in a geographically distributed system must be taken into consideration. Finally, the use of this system to simulate a large, complex and challenging simulation model will illustrate the potential of this system, especially when optimization via simulation is the ultimate goal.

ACKOWLEDGMENT

This research is supported by the Thailand Research Fund Grant Number MRG4980121.

REFERENCES

[1] B. L. Nelson, "50th anniversary article stochastic simulation research in Management Science," *Management Science* vol. 50, no. 7, pp. 855–868, 2004.

[2] M. C. Fu, "Optimization for simulation: theory vs. practice," *INFORMS Journal on Computing*, vol. 14 no. 3, pp. 192–215, 2002.

[3] M.C. Fu, F. W. Glover, and J. April, "Simulation optimization: A review, new developments, and applications," in *Proceedings of the 2005 Winter Simulation Conference, ed.*, M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. 2002, pp.184–191.

[4] S. Andradóttir, "An overview of simulation optimization via random search," in *Handbooks in Operations Research and Management Science: Simulation, ed.*, S. Henderson and B. L. Nelson. Amsterdam: North Holland, 2006, vol. 13, pp. 617–632.

[5] S. Chaisiri, J. Pichitlamken, P. Uthayopas, T. Rojanapanpat S. Phakhawirotkul, and T. Vorakosit, "Applying web services and windows clustering for high volume risk analysis," in *Proceedings of the 8th International Conference on High Performance Computing in Asia Pacific Region (HPC ASIA), ed.*, J. Fan and J. Lee. Beijing, China: Institute of Computer Technology, Chinese Academy of Sciences, 2005.

[6] H. He, "What is service-oriented architecture," Available via <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html> [accessed January 3, 2007].

[7] R. Duncan, "A survey of parallel computer architectures," *IEEE Computer*, 1990, pp. 5–16.

[8] SP Parallel Programming Workshop 2003. "Parallel programming introduction," Available via <www.mhpcc.edu/training/workshop/parallel_intro/MAIN.html> [accessed January 3, 2007].

[9] M. S. Snir, Otto, S. Huss-Lederman, D.Walker, and J. Dongarra, *1998. MPI: The complete reference volume 1: The MPI core*. 4h ed. Cambridge, Massachusetts: MIT Press, 1998.

[10] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir, *MPI: The complete reference volume 2: The MPI-2 extensions*. Cambridge, Massachusetts: MIT Press. 1998.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel virtual machine: A user's guide and tutorial for network parallel computing*. Cambridge, Massachusetts: MIT Press, 1994.

[12] J. Grosling, B. Joy, G. Steele, and G. Bracha, *The Java language specification*. 3d ed. Addison–Wesley Professional, 2005.

[13] D. S. Platt, *Introducing Microsoft .NET*. 3d ed. Microsoft Press, 2003.

[14] D. Spector, "Linux in the enterprise at LWE 2003." Available via <www.linuxdevcenter.com/pub/a/linux/2003/02/13/enterprise.html> [accessed January 3, 2007].

[15] W. Gropp, E. Lusk, and T. Sterling, *Beowolf cluster computing with Linux*. 2d ed. Cambridge, Massachusetts: MIT Press, 2003.

[16] J. D. Sloan, *High performance linux clusters: With OSCAR, Rocks, openMosix, and MPI*. O'Reilly & Associates. 2004.

[17] T. Erl, *Service-oriented architecture: A field guide to integrating XML and Web Services*. Prentice Hall Ptr., 2004.

[18] C. T. Ragsdale, *Spreadsheet modeling & decision analysis: A practical introduction to management science*. 3d ed. Mason, Ohio: South-Western. 2004.

[19] A. Tanenbaum, *Modern operating systems*. 2d ed. Prentice Hall. 2001.

Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Moench, O. Rose, eds.

HIGH PERFORMANCE SPREADSHEET SIMULATION ON A DESKTOP GRID

Juta Pichitlamken

Department of Industrial Engineering
Faculty of Engineering, Kasetsart University
Bangkok, 10900, THAILAND

Supasit Kajkamhaeng
Putchong Uthayopas

High Performance Computing and Networking Center
Faculty of Engineering, Kasetsart University
Bangkok, 10900, THAILAND

ABSTRACT

We present a proof-of-concept prototype for high performance spreadsheet simulation called S3. Our goal is to provide a user-friendly, yet computationally powerful simulation environment for end users. Our approach is to add power of parallel computing on Windows-based desktop grid into popular Excel models. We show that, by using standard Web Services and Service-Oriented Architecture (SOA), one can build a fast and efficient system on a desktop grid for simulation. The complexity of parallelism can be hidden from users through a well-defined computation template. This work also demonstrates that a massive computing power can be harvested by linking off-the-shelf office PCs into a desktop grid for simulation. The experimental results show that the prototype system is highly scalable. In the best case, the execution time can be reduced 13.6 times using 16 desktop PCs; the simulation time is dramatically reduced from 200 minutes to 14 minutes.

1 INTRODUCTION

Spreadsheet is a widely used computer application because of its versatility and ease of use in calculation or modeling a problem. Currently, the most popular spreadsheet program is Microsoft Excel on Windows since Windows shared more than 90% of the client operating system market (as of 2004) (Legard 2004). In fact, many textbooks use Excel as a calculation tool, e.g., Ragsdale (2004) and Stevenson and Ozgur (2007) which prove that Excel is sufficiently applicable to many types of analysis.

What-if analysis is used to assess how sensitive outputs are to changes in input values; e.g., how much total cost would increase if the project is delayed by x days. The analysis is inefficient if one parameter is changed at a time. Stochastic simulation is a widely-used technique for sensitivity analysis as it allows users to explore many more scenarios automatically. Applications of spreadsheet simulation include financial risk analysis (e.g., Paisittanand and Olson 2006) and operational risk analysis (e.g., Shariff et al. 2006). Seila (2006) and Seila, Ceric, and Tadikamalla (2003) provide a comprehensive introduction to spreadsheet simulation.

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 to solve this problem is to enhance Excel using a parallel/distributed computing. The idea is to break a large simulation problem into many small sub-problems that can be executed concurrently on multiple computers. Excel can be customized through user-defined Visual Basic for Applications (VBA) macros or add-ins that allow users to include their own functions; therefore, it is possible to modify Excel to seamlessly use parallel computing to speed up its execution of a large simulation problem.

In this paper, we build a **s**preadsheet **s**imulation **s**ystem (**S3**) that is easy to use and computationally fast by utilizing power of parallel computing on a Windows-based desktop grid. When simulation runs are faster, an analyst can execute a simulation model under multiple sets of decision variables, or they may want to find the best set of decision variables for the system of interest (*optimization via simulation*). S3 allows users to execute a spreadsheet simulation model under multiple set of integer-valued decision variables by specifying their upper and lower bounds. For each combination of decision variables, S3 returns sample means and standard errors. Users can then explore these outputs by sorting them and examine which set of decision variables give the top x% performance.

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 mod-

els or build their models without being aware of parallelism and infrastructure used. 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* which are built from commodity PCs, readily available in most organizations. 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, Kesselman, and Tuecke 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 desktop grids 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.

The second issue is more challenging. Typically, Excel evaluates formulas and display the results as values in the cells that contain the formulas (Ecklund 2007). A `.xls` file is an Excel *Workbook* which consists of *Worksheets*. 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) calls 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. We have a Worksheet template where a user specifies the number of inputs and their cell locations and similarly for outputs. When simulation runs are finished, users get outputs of each replication (if there is only one set of decision variables) or sample means and standard errors of each set of decision variables, i.e., a parametric sweep. Due to space limitation, we will only discuss the template and result display of the latter case.

This paper is organized as follows: We summarizes 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 direction in Section 5.

2 RELATED WORK

In this section, we summarize works that address the issue of achieving parallelism, especially for spreadsheet.

Condor is a widely-known tool for maximizing utilization of computing resources (Litzkow, Livny, and Mutka 1988). The Condor scheduling system identifies idle machines and schedules background jobs on them. When those machines are used for non-Condor jobs, the Condor job is terminated and transferred to another machine, with the last system state before termination.

ActiveSheets is an application that allow parallel evaluation of spreadsheets. User interface is Microsoft Excel. ActiveSheets require custom functions for parallel calculation which are done at backend computers. When the computation is finished, the results are displayed on a spreadsheet. In Abramson et al. (2001), the backend platform is managed by EnFuzion (`www.axceleon.com`) on a high performance computing (HPC) system such as computer clusters or grids. On the other hand, Abramson et al. (2004) uses NetSolve (Agrawal et al. 2003), a grid middleware.

Nadiminti et al. (2004) introduce *ExcelGrid*, an open-source .NET plug-in that uses Excel as a front-end to a grid and perform user-defined calculations on it. Mustafee et al. (2006) show how *WinGrid* can enable *Witness*—a commercial simulation package—perform simulation replications in parallel on enterprise grid (linked resources within the same organization, as opposed to the public grid). In this work, users do not build a simulation model on Witness directly but specifying model parameters through Excel. The simulation results are also displayed on Excel.

ActiveSheets, ExcelGrid and WinGrid use Excel mainly for user interface; they do not perform computing-intensive calculation on Excel, and they are not designed specifically for spreadsheet simulation. On the other hand, our S3 still exploits Excel calculation, and it aims specifically at spreadsheet simulation applications. Hence, no complex programming is needed to benefit from our approach.

Microsoft also offers a tool for creating Excel Services for running a parametric sweep on a Excel 2007 Windows Compute Cluster Server (CCS) 2003 (Microsoft Corp. 2008). Excel Services is an architecture that allows Excel calculation on servers and enable 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 spreadsheet simulation Excel add-ins also offer parallel versions: @Risk (`www.palisade.com`) has RiskAccelerator$^{\text{TM}}$, and Crystal Ball (`www.crystalball.com`) provides Crystall Ball Turbo$^{\text{TM}}$.

3 DESIGN AND ARCHITECTURE

We first describe a S3 spreadsheet simulation model. Then we provide details on system design and implementation.

3.1 Spreadsheet Simulation Models

By design, each simulation replication (i.e., runs) of the same model yield outputs that are independent and identically distributed (i.i.d.); therefore, we can achieve parallelism by assigning simulation replications that use different set of random inputs to each compute node, i.e., parallelism is achieved via *domain decomposition* (Quinn 2003). This technique relies on the execution of the same spreadsheet on multiple computers using different data set for each one. Once the calculation 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. 2005 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 Henderson and Nelson 2006 for brief summaries on RNGs and random variate generation).

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.

We illustrate our approach via an example (more details in Section 4.1): We estimate value-at-risk (VaR) of stock portfolios by first simulating daily stock prices. In doing so, we need normally distributed random variables which in turn require uniform (0,1) random numbers. We fix the names of the following two Worksheets:

- `Model` contains calculation.
- `SimRun` is a template where a user specifies details on uniform(0,1) random numbers (number of rows and columns and their locations), outputs (number of outputs and their cell locations), and decision variables (number of decision variables, their location and their respective lower and upper bounds). See Figure 7. The location of some of these cells are fixed, but some cell locations depend on locations of other cells. `SimRun` also holds uniform(0,1) random numbers that Worksheet `Model` needs. The RNG that our compute nodes use is Mersenne Twister ("mt19937") where we adapt to C# from C code in the GNU Scientific Library (Galassi et al. 2006). S3 knows the details about a simulation model through this worksheet.

In this VaR example, we have three simulation outputs that are estimated from multiple i.i.d. trials in Worksheet `Model`: VaR, average portfolio values on the next day and average profit/loss. S3 knows that they are outputs because we link them to Worksheet `SimRun`. Users can also execute simulation runs under multiple set of decision variables (DVs) by specifying the upper and lower bounds of each DV in Worksheet `SimRun` (see Figure 7). Currently, we only allow integer values with step size of 1. In this example, our DVs are the number of lots (one lot consists of 100 stocks) of each stock in our portfolio. DVs are specified on Worksheet `SimRun`, and they are used in Worksheet `Model` for calculation.

Once the calculation is completed, a user gets result on another Worksheet called `Output` (see Figure 8). For each combination of DVs in the specified range, sample means and standard errors are provided. (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 Design and Implementation

We first explain the execution steps of S3 and discuss the details about system architecture.

The S3 system architecture is shown in Figure 9. The system is consisted of four main components:

1. *Users* upload Excel simulation models and download Excel output files when jobs are completed.
2. *Manager* is responsible for resource management (book keeping of status of workers), 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 Manager even if there are other jobs running on them.
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.

The execution steps of S3 can be explained as follows (the number below correspond to ones in Figure 1).

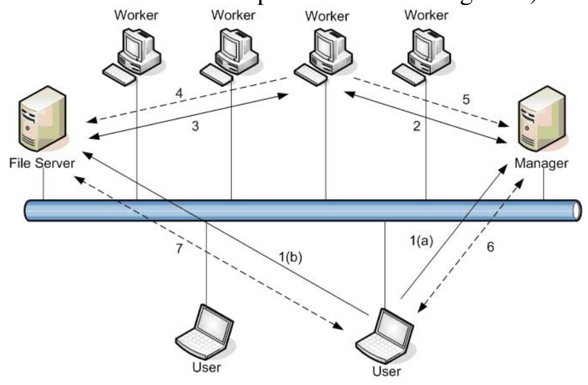


Figure 1: System configuration.

1. A user creates a simulation model in Excel which has our add-in that connects to Manager through Web Services. Figure 2 shows the dialog box for job submission. Job description (such as the number of replications, number of workers and a seed number) is 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 maybe 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.
5. Then Worker updates his status with Manager.
6. A user can check status of his submitted jobs through Manager (see Figure 3).
7. When user's job is completed, he downloads his output Excel file from File Server.

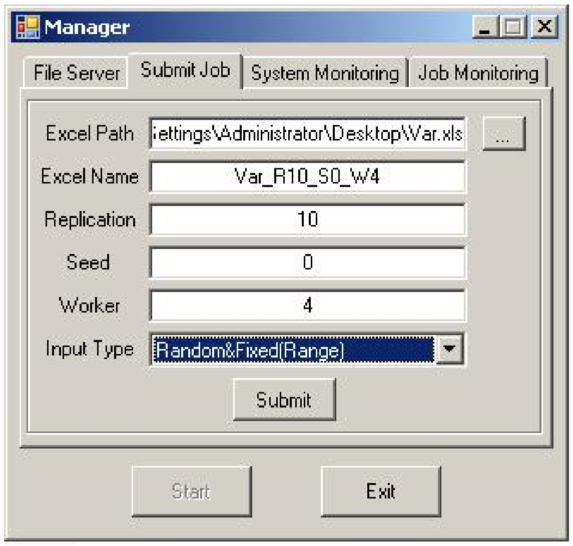
Figure 8 shows an example of output display.

4 EXPERIMENTAL RESULTS

We first describe our test problem and the experimental setups, followed by experimental results and discussion.

4.1 Value-at-Risk Test Problem

VaR is a risk metric that probabilistically describe market risks. Let L be investment profit or loss. Given some confidence level $\alpha \in (0,1)$, the VaR of the portfolio at the confidence level α is given by the smallest number ℓ such that $\Pr\{L > \ell\} = 1 - \alpha$ (Jorion 2001). For a given



Excel Path: ıettings\Administrator\Desktop\Var.xls
Excel Name: Var_R10_S0_W4
Replication: 10
Seed: 0
Worker: 4
Input Type: Random&Fixed(Range)

Figure 2: Dialog box when a user requests simulation runs.

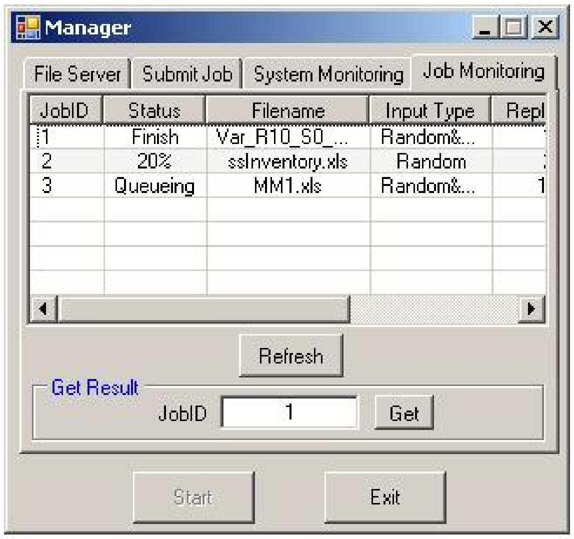
combination of stocks, we simulate a value-at-risk over one-day period. We assume that stock prices follow a Brownian motion process. The calculation steps are (Hull 2003):

1. Compute the value of portfolio today using the current stock prices. Let S_{it} be the price of stock i on day t, and x_i is the number of stock i in the portfolio, $i = 1, 2, \ldots, n$. The value of portfolio on day t is $P_t = \sum_i x_i S_{it}$.
2. Sample stock returns, $\mathbf{r} = [r_1, r_2, \ldots, r_n]'$, from the multivariate normal distribution whose mean is zero and the covariance matrix is estimated from historical data.
3. Use r_i to determine the stock price on day $t+1$: $S_{i(t+1)} = S_{it}e^{r_i}$. Revalue the portfolio value on day $t+1$, $P_{t+1} = \sum_i x_i S_{i(t+1)}$.
4. Compute $\delta P = P_{t+1} - P_t$.
5. Repeat steps 2 to 4 m times to get samples of δP.

We estimate the α% VaR as the α percentiles of m simulated values of δP. We use $m = 200$ and $n = 10$ stocks that are traded in the Stock Exchange of Thailand. The covariance matrix of stock returns are estimated from 50 trading days, from December 7, 2007 to February 20, 2008.

4.2 Experimental Setups

The experiment is done on a 19-PC system. The user's system is an AMD Athlon XP 1GHz system with 512MB RAM installing Windows XP. Both the manager and file server system are an AMD Athlon XP 2500+ system with 512MB RAM installing Windows Server 2003. The rest of computing nodes are an Intel Celeron 2.53 GHz system



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

Figure 3: Dialog box when a user checks job status.

with 512MB RAM installing Windows XP. All machines are connected together using 100Mbps Fast Ethernet switch. In this work, all the softwares are developed with Visual Studio 2005 and Visual Studio Tool for Office (VSTO). The manager uses the FCFS (first come, first served) algorithm for job assignments.

We vary the number of combinations of decision variables (i.e., problem size) by changing the upper and lower bounds of each DVs, and the number of workers used. Each combination of decision variables gets 10 simulation replications. The performance measure is the computational time to complete the simulation runs. The test is done at 256, 512, 1024, 2048, 4096 combinations of decision variables. The number of workers used are 1, 2, 4, 8, and 16 workers.

4.3 Experimental Results

The runtime results are shown in Figure 4. We see that the run time decreases when number of processing nodes increases. This is due to the distribution of the processing tasks to multiple computing nodes simultaneously. However, benefits of increasing the number of workers diminish as the number of workers increases, e.g., the runtime decreases sharply when we include the second worker, but the benefit declines as we go from 8 to 16 workers.

We also consider a performance measure called *speedup* which is defined as a ratio between sequential runtime (runtime on one worker) and parallel runtime (runtime on multiple workers). Parallel run time consists of computation

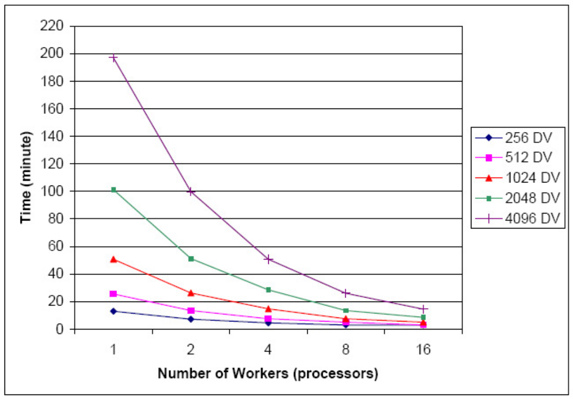


Figure 4: Runtime on the desktop grid when the problem sizes and the number of workers vary.

time and communication overhead. Thus,

$$\text{Speedup} = \frac{T_{\text{Seq}}}{T_{\text{Parallel}}} = \frac{T_{\text{Seq}}}{T_{\text{comp}} + T_{\text{comm}}}. \quad (1)$$

Speedup shows how many times faster the execution is when parallel computing is used; if we use n identical processors, the ideal speedup is n, i.e., when $T_{\text{comm}} = 0$, $T_{\text{comp}} = T_{\text{seq}}/n$ in (1). The actual speedup is lower due to communication overhead, which increases with the number of processors and the problem size. In our experiment, the communication overhead is mostly due to preparing communication channels between servers and workers, rather than in uploading or downloading files. Thus, for a given the number of processors, our communication overhead are relatively close across all problem sizes, and speedup for large problems is higher than for small problems. (that we consider).

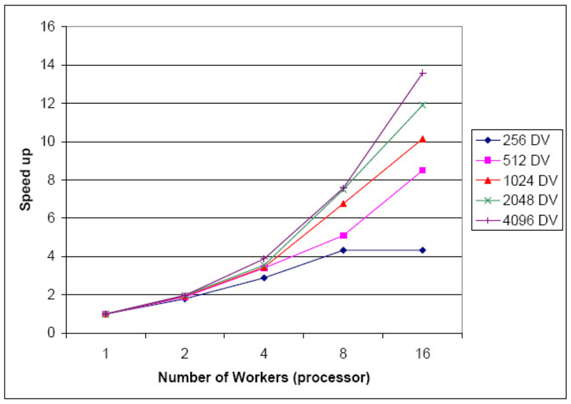


Figure 5: Speedup on the desktop grid when the problem sizes and the number of workers vary.

In Figure 5, we observe the following results:

- Speed up increases at a faster rate when the number of computing Service nodes are added. Maximum speedup gained in this experiment is 13.6; therefore, the application runs 13-14 times faster for the system of only 16 nodes.
- Speedup for large problems is higher than for small problems. The reason is that when more computing workload is available, the fraction of computing overhead (e.g., in load balancing and communication) to the computing workload is lower. Thus, our proposed system will work even better for large and complex problems. Note the seemingly unusual behavior of the results for 256DVs, where the runtime for 8 and 16 processors are approximately 3 minutes. This is because the communication overhead is much larger than the parallel computational time, and this overhead is also close to the sequential run time. As a result, speedup is non-increasing (see (1)).

A fraction of runtime used as system overhead can be assessed through a ratio called *efficiency*. The efficiency of parallel computation is speedup divided by number of workers. Efficiency indicates the effectiveness that our computing systems are utilized to solve the problem. Due to communication overhead, efficiency is between 0 and 1 (100%), where being closer to one is desirable. Figure 6 shows 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 number of workers increases, the communication overhead increases as well. Thus, the efficiency is decreasing accordingly, and at a higher rate. In addition, 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 80% (0.8) for 16 nodes with the problem size of 4096 DVs.

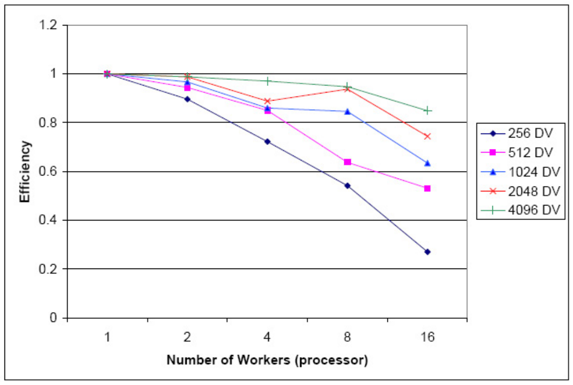


Figure 6: Efficiency on the desktop grid when the problem sizes and the number of workers vary.

5 FUTURE WORK

In this paper, we propose an architecture that allows spreadsheet simulation to use Windows-based desktop grids to accelerate the execution speed. Our approach is different in that we base almost all computations on Excel spreadsheets (except for random number generations). Thus, no complex programming is needed. We also show that the complexity of parallel computing can be mostly hidden from users through well-designed computation templates in Excel. With these templates, users can have a flexibility of modeling a simulation problem while enjoying massive computing power. From the experiments, we show that runtime can be reduced from 200 minutes to about 14 minutes. This speedup can make a huge difference in how a user analyzes problems; with desktop grids, they are able to consider many scenarios simultaneously or even to optimize a sizable model, thus gaining greater insights on the problem at hand.

Our work can be enhanced in many ways. More templates can be added for broader classes of problems. More transparency can be built so users are not aware of the manager existence by adding automatic job submission into Excel. For a longer term execution, some of the fault handling mechanism should be added to make it easy to use the system in a less reliable IT environment.

ACKNOWLEDGMENTS

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. We would also like to thank the WSC referees for careful reading and helpful comments.

REFERENCES

Abramson, D., J. Dongarra, E. Meek, P. Roe, and Z. Shi. 2004. Simplified grid computing through spreadsheets and NetSolve. In *Proceedings of the Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region (HPCAsia'04)*. IEEE Computer Society.

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

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

Anderson, D. P. 2004. BOINC: A system for public-resource computing and storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*. Available via <http://boinc.berkeley.edu/grid_paper_04.pdf> [accessed March 11, 2008].

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2005. *Discrete-event system simulation*. 4th ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Ecklund, P. 2007. Notes on excel calculations. Available via <http://faculty.fuqua.duke.edu/ pecklund/ExcelReview/ExcelFormulasReview.pdf> [accessed March 11, 2008].

Foster, I., C. Kesselman, and S. Tuecke. 2001. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications* 15 (3): 200–244.

Galassi, M., J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. 2006. *GNU scientific library reference manual*. 2nd ed. Network Theory Ltd.

Henderson, S. G., and B. L. Nelson. 2006. *Handbooks in operations research and management science, volume 13: Simulation*. North Holland: Springer-Verlag.

Hull, J. C. 2003. *Options, futures & other derivatives*. 5th ed. Prentice Hall.

Jorion, P. 2001. *Value at risk: The new benchmark for managing financial risk*. 2nd ed. McGraw-Hill Trade.

Knusel, L. 2005. On the accuracy of statistical distributions in Microsoft Excel 2003. *Computational Statistics and Data Analysis* 48 (3): 445–449.

Legard, D. 2004. IDC: Consolidation to windows won't happen. Available via <http://www.linuxworld.com.au/index.php/id;940707233;fp;2;fpid;1> [accessed March 11, 2008].

Litzkow, M. J., M. Livny, and M. W. Mutka. 1988. Condor-a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, 104–111.

McCullough, B., and B. Wilson. 2005. On the accuracy of statistical procedures in Microsoft Excel 2003. *Computational Statistics and Data Analysis* 49 (4): 1244–1252.

Microsoft Corp. 2008. Microsoft Excel running on Microsoft compute cluster. Available via <http://msdn.microsoft.com/en-us/library/bb463068.aspx> [accessed July 7, 2008].

Mustafee, N., A. Alstad, B. Larsen, S. J. E. Taylor, and J. Ladbrook. 2006. Grid-enabling FIRST: Speeding up simulation applications using WinGrid. In *Proceedings of the 10th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2006)*, 157–164.

Nadiminti, K., Y.-F. Chiu, N. Teoh, A. Luter, S. Venugopal, and R. Buyya. 2004. ExcelGrid: A .NET plug-in for outsourcing Excel spreadsheet workload to enterprise and global grids. In *Proceedings of the 12th International Conference on Advanced Computing and Communication (ADCOM 2004)*. Available via <http://www.gridbus.org/papers/ExcelGrid.pdf> [accessed March 11, 2008].

Paisittanand, S., and D. L. Olson. 2006. A simulation study of IT outsourcing in the credit card business. *European Journal of Operational Research* 175 (2): 1248–1261.

Quinn, M. 2003. *Parallel programming in C with MPI and OpenMP*. McGraw-Hill.

Ragsdale, C. T. 2004. *Spreadsheet modeling & decision analysis*. 4th ed. Mason, Ohio: South-Western (Thomson Learning).

Seila, A. F. 2006. Spreadsheet simulation. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 11–18. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Seila, A. F., V. Ceric, and P. Tadikamalla. 2003. *Applied simulation modeling*. Thomson Learning.

Shariff, A. M., R. Rusli, C. T. Leong, V. Radhakrishnan, and A. Buang. 2006. Inherent safety tool for explosion consequences study. *Journal of Loss Prevention in the Process Industries* 19 (5): 409–418.

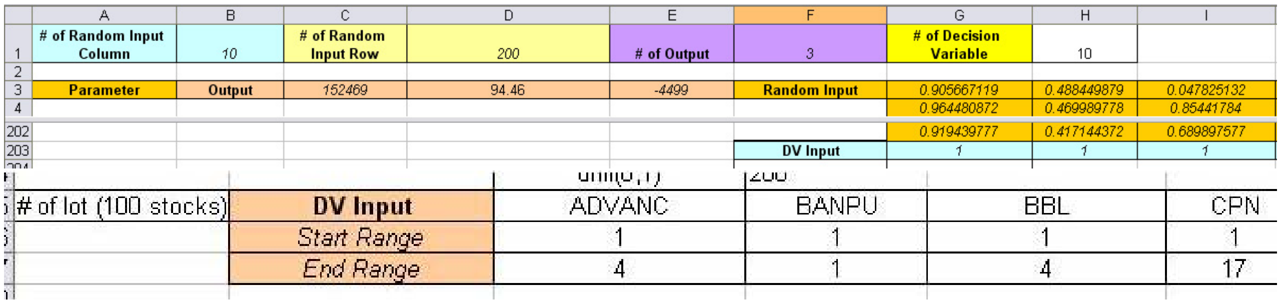
Stevenson, W. J., and C. Ozgur. 2007. *Introduction to management science with spreadsheets*. New York, NY: McGraw-Hill/Irwin.

AUTHOR BIOGRAPHIES

JUTA PICHITLAMKEN is an Assistant Professor in the Department of Industrial Engineering, Kasetsart University. Her research interests include ranking and selection procedures, simulation optimization, spreadsheet simulation, and stochastic processes. Her e-mail is <juta.p@ku.ac.th>.

SUPASIT KAJKAMHAENG is a Master student in High Performance Computing and Networking Center, Kasetsart University. His recent work has involved Cluster and Grid Computing. His email is <kurata_sk@hotmail.com>.

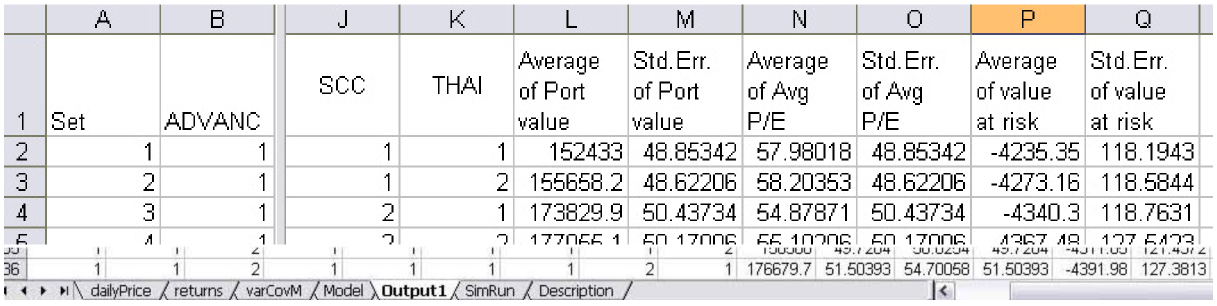
PUTCHONG UTHAYOPAS is an Assistant Professor in Department of Computer Engineering, Kasertsart University. His research interests include parallel/distributed computing, cluster and grid computing and parallel software tools. His email is <pu@ku.ac.th>.



	A	B	C	D	E	F	G	H	I
1	# of Random Input Column	10	# of Random Input Row	200	# of Output	3	# of Decision Variable	10	
2									
3	Parameter	Output	152469	94.46	-4499	Random Input	0.905667119	0.488449879	0.047825132
4							0.964480872	0.469989778	0.85441784
202							0.919439777	0.417144372	0.689897577
203						DV Input	1	1	1

# of lot (100 stocks)	DV Input	ADVANC	BANPU	BBL	CPN
	Start Range	1	1	1	1
	End Range	4	1	4	17

Figure 7: Screenshot of Worksheet `SimRun`.



	A	B	J	K	L	M	N	O	P	Q
1	Set	ADVANC	SCC	THAI	Average of Port value	Std.Err. of Port value	Average of Avg P/E	Std.Err. of Avg P/E	Average of value at risk	Std.Err. of value at risk
2	1	1	1	1	152433	48.85342	57.98018	48.85342	-4235.35	118.1943
3	2	1	1	2	155658.2	48.62206	58.20353	48.62206	-4273.16	118.5844
4	3	1	2	1	173829.9	50.43734	54.87871	50.43734	-4340.3	118.7631

36	1	1	2	1	1	1	1	2	1	176679.7	51.50393	54.70058	51.50393	-4391.98	127.3813

dailyPrice / returns / varCovM / Model / **Output1** / SimRun / Description

Figure 8: Screenshot of Worksheet `Output`.

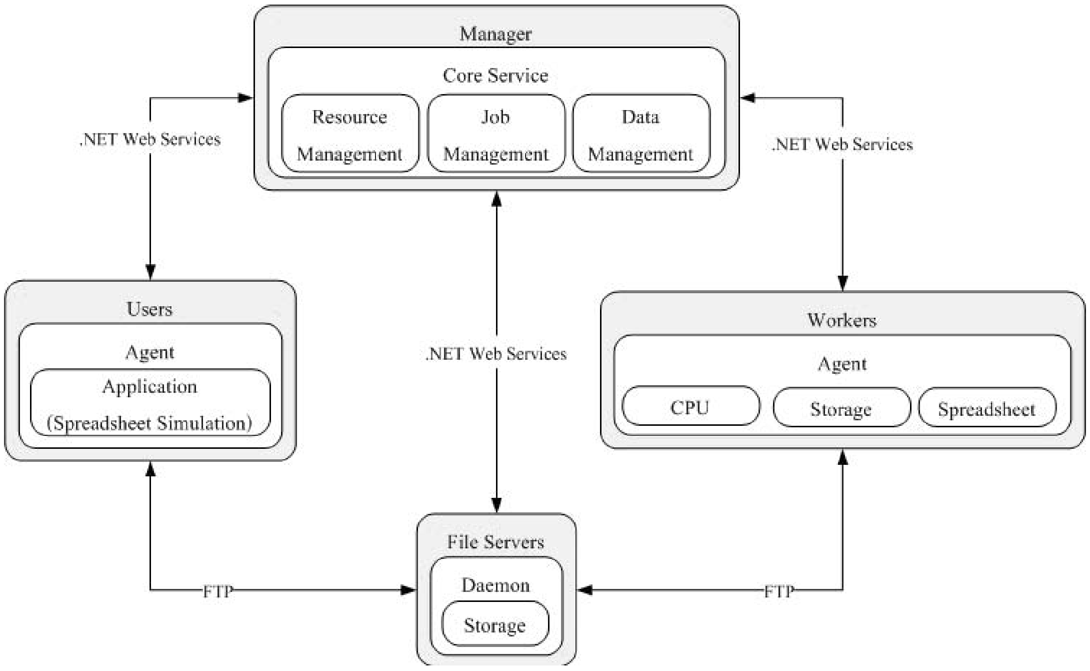


Figure 9: S3 architecture.