Design and Implementation of a Framework for .NET-based Utility Computing Infrastructure

Thanapol Rojanapanpat* and Putchong Uthayopas

ABSTRACT

Future organizations must handle a very large and complex IT infrastructure that consists of very diverge and highly heterogeneous computing systems. Moreover, the future generation applications must access services and resources regardless of the geographical location, access methods, and domain of authorization. In order to meet these challenging requirements, a very high degree of virtualization has to be implemented using a smart middleware. This is a very challenging problem for both theory and practice.

This paper presents a new framework called OpenUCI (Open Utility Computing Infrastructure). The OpenUCI project aims to explore the innovative design of scalable and flexible software infrastructure that manages large scale heterogeneous distributed system ranging from large Server, PC, and Mobile Devices. OpenUCI exploits a well established technology such as Grid, Web services and .NET technology to build a virtualized and unify access to resources. Basic services that need to be presented will be discussed. The prototype system has been implemented along with the prototype financial engineering application. The results are presented along with the discussion of the experiences learned. With OpenUCI, users can easily harness computing and storage of large distributed system.

Key words: utility computing, .NET technology, web services

INTRODUCTION

The competition in business causes organizations to be ready to handle a large amount of demand of users, which need more high performance computing system. It is a risk for the small and medium organizations to invest in the high performance computing system, because they have to pay for the system maintenance cost. There are two solutions. Firstly they can outsource the computing power. The other solution is to create the supercomputing system by utilizing the already existing personal computers (PC) in their company. Building a supercomputing system from personal computers or desktop PCs now is not an imagination, because the speed and performance of PCs has been increasing as well as the speed and bandwidth of network. From this advantage, it emerges many new computing systems; one of them is the utility computing system.

Utility computing (Eilam *et al.*, 2004) is a computing model that involves the use of many diverge technology such as grid computing (Foster *et al.*, 2002) and autonomic computing (Ganek and Corbi, 2003). Utility computing system focuses on the creating of virtual computing environment

High Performance Computing and Networking Center, Faculty of Engineering, Kasetsart University, Bangkok 10900, Thailand,
*   Corresponding author, e-mail: thanapolr@hpcnc.cpe.ku.ac.th, pu@ku.ac.th

Received date : 03/10/06   Accepted date : 25/12/06

which dynamically and automatically virtualizes, provisions and manages resources and services on users' demand. The major benefits of utility computing are

- better utilization – the resources in utility computing system can be shared and used in efficient ways,
- more flexibility – utility computing system provides flexibility in the creation of the dynamic computing environment which can automatically increase or decrease the computing resources corresponding to users' demand, and
- lower total cost of ownership – utility computing can provide IT and business process outsourcing which help reducing cost of investing in resources such as hardware assets, maintenance cost, training cost, etc.

The design and building of utility computing infrastructure is still a complex and challenging task. Figure 1 shows the concept of resources virtualization and resources provisioning in a modern IT infrastructure. Utility computing system must consist of a way to provide both features. Firstly, resources virtualization is a feature of system that can make resources transparent to application. Since the resources that we use for build a utility computing infrastructure are PCs, these systems have a high dynamism e.g. resources can be available and unavailable from time to time. The utility computing system must have mechanisms for collecting resources and monitoring its status. Furthermore, resource virtualization should have mechanisms for discovering and accessing resources. Secondly, resources provisioning is a feature of a system to perform an on-demand resources allocation to application. A good utility computing system must have mechanism for creating the automatic adjustable virtual computing environments which consist of hardware resources and utility services in order to keep responsiveness when users' demand increases. Moreover, the utility computing system must provide friendly-used interfaces to user, which may be business manager for accessing and managing this virtual computing environment. These interfaces can be Windows applications, Web applications, command line, and API.

Most utility computing systems are based on current distributed system technology. Thierry (2006) provides a good survey of platform technology that is available. There are three commonly used distributed systems and technologies. The first one is *distributed information system* that focuses on sharing knowledge such as the web. Secondly, a *distributed storage system* for sharing data such as peer-to-peer files sharing. Finally, *distributed computing or metacomputing* (Smarr and Catlett, 1992) frameworks for sharing computing power. The systems that are classified in this area and related to OpenUCI framework are the followings.

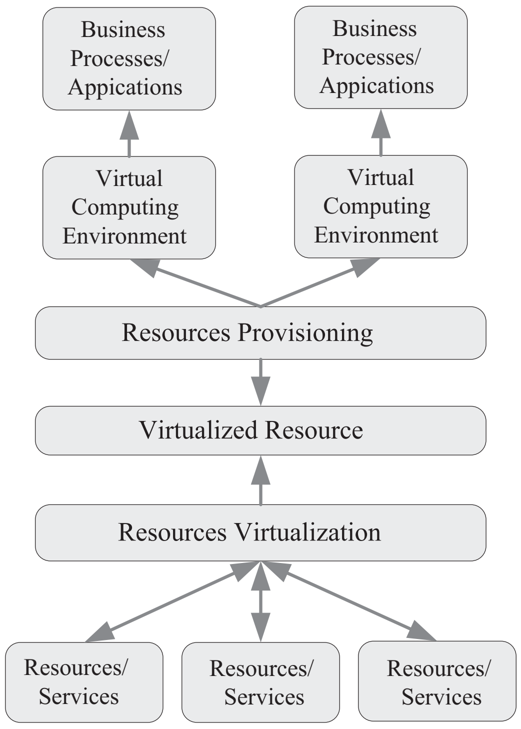

Figure 1 The relationship of resource viortualization and resource provisioning.

Grid computing (Foster *et al.*, 2002) focuses on integrating geographically distributed resources into a unified system. Grid computing provides concept of Virtual Organization (VO) which is an integrated resources shared by real organizations, and it also has a well-defined architecture, services and protocols such as resource discovery, job submission, system monitoring and accounting, which are good patterns for designing and developing the utility computing system. The most well-known project in this area is Globus (The Globus Alliance, 2005a).

Peer-to-Peer (P2P) computing is a class of applications that takes advantage of resources such as storage, CPU cycles, and content that are available on the Internet. There are two major categories of P2P system, P2P networking (file sharing) and P2P computing (CPU sharing), The P2P networking is a communication model in which each node (*peer*) has the same capabilities and either node can directly initiate a communication session. The P2P computing is a processing power sharing rather than a files sharing.

Volunteer computing (Sarmenta, 2001) focuses on making computers to be a part of metacomputer dynamically when computing power is available. The topology of volunteer computing is usually similar to the third generation of peer-to-peer computing. The peer can be both client, who submits jobs to server (super-peer), and can be worker who dedicates itself to execute jobs. This includes system such as SETI@home (Anderson *et al.*, 2002), Bayanihan (Sarmenta *et al.*, 2002), and Alchemi (Luther, 2005).

In this paper, we present a design and implementation of a framework called OpenUCI (Open Utility Computing Infrastructure) which is for constructing the utility computing infrastructure from Windows-based personal computers, because the most of computers in the organization are Windows-based operating system and the most of users are familiar to Windows. To solve the resource virtualization and resource provisioning problems, OpenUCI framework provides many services such as resource collecting, resource monitoring, resource discovering, resource invocation, and etc. In addition, we use Microsoft's .NET technology for implementing the OpenUCI system because it provides a powerful and comfortable development environment and it also provides ASP.NET Web service, a standard way for communication between systems. So, we can ensure that all OpenUCI's components can work together and can communicate to other systems seamlessly.

MATERIALS AND METHODS

1. Hardware and software requirements

This paper develops and tests a framework on Windows-based system. The computers used in this development comprise one manager node, 32 worker nodes, and one user node. All nodes are connected with Fast Ethernet switch. The system configuration is shown in Figure 2.

The software for developing and testing the framework is as follows:

- Microsoft Windows Server 2003
- Microsoft Windows XP Professional
- .NET framework redistributed 1.1 and 2.0
- Microsoft Visual Studio .NET 2003 and 2005

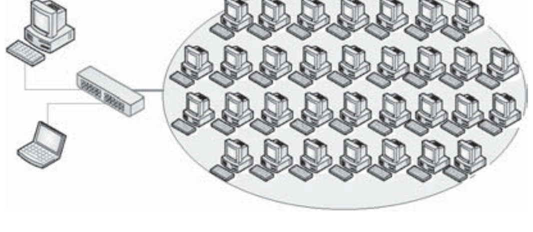

Figure 2 The windows cluster.

2. Framework architecture and components

In this paper, utility service is a function provided by any computers. The utility service must depand on the Service Oriented Architecture (SOA) technology such as .NET web services, and Grid services. The example of utility service is such web service for calculating risk of trading stock (VaR) (Rojanapanpat *et al.*, 2005). The resource is an entity shared by a computer and can be computing power (CPU), storage, files and utility services.

According to the utility computing system development problems mentioned before, *Resources Virtualization* and *Resources Provisioning,* the proposed framework, OpenUCI, must be designed to solve these problems.

To deal with *Resources Virtualization* problem, OpenUCI must have mechanism to support the *dynamism*, *heterogeneity*, *scalability*, *interoperability* of resources. The mechanisms are such *resource collecting* for gathering resources and track its status, *resource discovery* used to find and select the resources, *resource accessing* which defines a unite way to use and interoperate resources and etc.

In the *Resources Provisioning* problem, OpenUCI must provide mechanisms for *creating virtual computing environments* that can be automatically adjustable depending on demand of users. Moreover, OpenUCI must provide *user-friendly interfaces and tools* using OpenUCI system and accessay resources to users.

The architecture of the OpenUCI framework is shown in Figure 3. There are four layers of the OpenUCI framework, i.e. resources, .NET platform, core services, and applications.

2.1 Resources layer

Resources layer is the layer of shared resources distributed on the network. The shared resources consist of CPU, storage and utility services.

2.2 .NET platform layer

.NET platform layer provides a runtime environment, .NET framework, which OpenUCI system relies on. This layer also provides technologies for implementing OpenUCI system, and sharing resources. These technologies are .NET web services, .NET remoting and WSRF.NET. The resources can be shared via these technologies.

2.3 Core layer

This layer provides a set of necessary services for building the utility computing infrastructure and supporting the basic functions of the application running on the utility computing infrastructure. The core services are classified into two groups according to our requirements.

The core services that solve the resources virtualization problem consist of resource management service, data management service and execution management service.

1. Resources Management Service (RMS) is responsible for gathering resources distributed on the network and tracking the existence and status of resources. Moreover, RMS also provides mechanisms for resource discovery, resource reservation and etc.

2. Data Management Service (DMS) is responsible for transferring files and sharing files

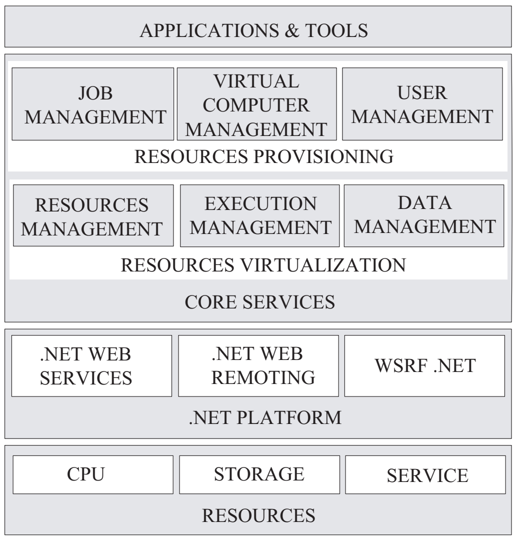

Figure 3 The OpenUCI architecture.

among computers in the OpenUCI system.

3. Execution Management Service (EMS) is used to start and controls processes. Furthermore, EMS also supports the invocation of web and grid service jobs.

The core services that address the resources provisioning problem consist of user management service, virtual computer management service and job management service.

1. User Management service (UMS) handles authentication, authorization, accounting and users profiles.

2. Virtual Computer Management Service (VCMS) is used for managing and controlling the virtual computing environment created by users.

3. Job Management Service (JMS) is used for creating jobs and supporting job submission from users. JMS also provides job queuing and scheduling mechanisms.

2.4 Applications and tools layer

Applications and tools layer is the layer of user applications developed for using facilities of OpenUCI system. OpenUCI system also provides basic command-line tools and web application interfaces for login, logout, virtual computer creation, resources discovering, job submission and etc.

There are three main components in OpenUCI system as shown in Figure 4.

1. Manager is a computer that provides core services used for managing shared resources and supporting incoming requests of users.

2. Workers are computers that share its' resources such as computing power, files, storage and utility services. There are two worker types in the OpenUCI system, dedicated and non-dedicated workers. Dedicated workers are always online and cannot reject jobs assigned by managers. For non-dedicated workers, they can be online or offline all the time and they will request for a job and execute it when they are not busy.

3. Users are the people who need to access resources. They can discover resources, create job, submit job, download and upload files and any services provided by managers.

RESULTS AND DISCUSSION

1. Proof of concept application

Currently, the high performance computing is widely needed and not limited to the computer research field anymore. The financial engineering (FE) is a field that requires the high computing power because it has to handle and analyze a large amount of data in order to reduce or keep turn around time constantly as number of users increased. We evaluated the performance of OpenUCI system by applying the existing financial engineering application named Value-at-

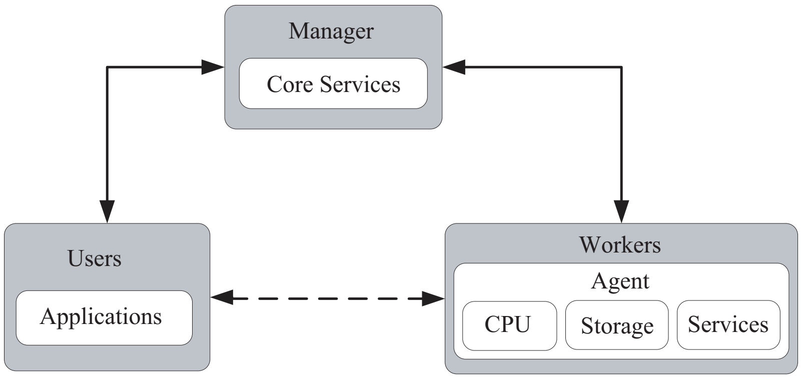

Figure 4 The interaction of manager, worker and user.

Risk (VaR) calculation which was implemented in .NET web services. The VaR measures the maximum loss money which may be occurred in portfolio at a given time horizon (time of holding portfolio) and at a given level of confidence. The formula for calculating VaR has high complexity. Then, we will show the general form of formula.

$$VaR = -Vp^* (\mu p - Q^* \sigma p)$$

The V_p is the portfolio value, and the μp and the σp are the expected return and the standard deviation, respectively. The Q is the quantile value of %confidence level. For example, the 99% confidence level gives ~2.326 quantile value and the 95% confidence level gives ~1.645 quantile value.

In this test, we used the VaR calculation web service as a utility service of OpenUCI system which was installed to all worker machines and then we developed VaR client program with Microsoft Excel. The VaR Excel program uses the OpenUCI API to connect to manager, discover VaR web services and then invoke them.

2. Test configuration

The topology of test system is shown in Figure 2. The software that was installed on each machine is shown in Table 1.

3. Test assumptions

- Each worker executes only one job at a time. Since the test application is a compute intensive application, the execution of more than one job on each worker will not give a better performance due to the overhead of task switching.
- The input data is already in the workers. This can be done by preloading fixed data and table to worker prior to the execution. Thus, the communication can be minimized which yield a better performance for the system.

4. OpenUCI throughput test

We evaluated the throughput of OpenUCI by submitting jobs to OpenUCI system that has 1, 2, 4, 8, 16, and 32 workers, and the run times used for testing are changed from 10, 30, 60, 90, 120, 180, 240, and 300 seconds. Figure 5 shows the procedure of this testing.

1. The client application discover URLs of web service located on the worker nodes from the manager.
2. The manager runs the resource selection algorithm and returns the URLs of the chosen worker node to requested client application.
3. The client application uses the returned URLs for connecting and invoking web service on worker nodes. After that, the client application will wait until there are some available workers.
4. The worker node executes the service and then it returns a result to client application.
5. The client program invokes web service on an available worker

Table 1 Hardware and software configuration for testing OpenUCI system.

Machines	Hardware	Operating system	Software
1 Manager	AMD Athlon 2.0GHz, 512 MB RAM	Windows server 2003	OpenUCI Broker, MS SQL 2005 for OpenUCI database
32 Workers	Intel Celeron 2.53GHz, 512 MB RAM	Windows XP Professional	OpenUCI Worker, MS SQL 2005 Express for VaR database
1 User	Intel Pentium M 1.5GHz, 768 MB RAM	Windows server 2003	VaR client application

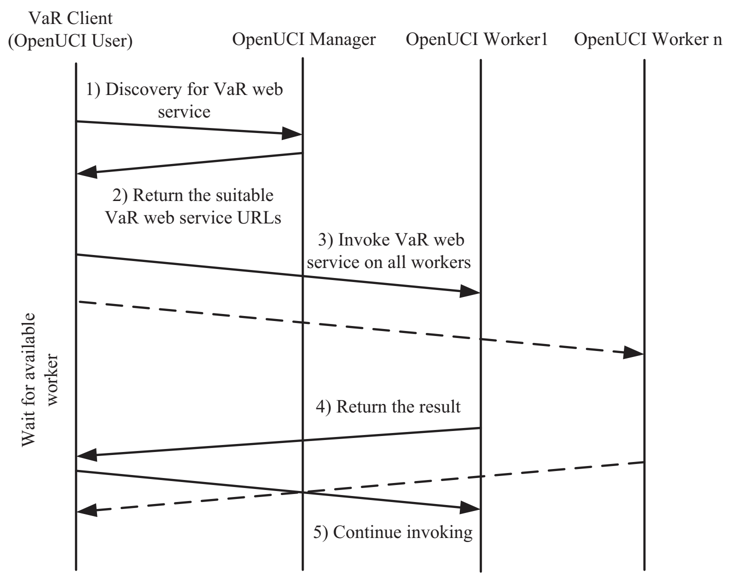
VaR Client
(OpenUCI User)
OpenUCI Manager
OpenUCI Worker1
OpenUCI Worker n
1) Discovery for VaR web service
2) Return the suitable VaR web service URLs
3) Invoke VaR web service on all workers
Wait for available worker
4) Return the result
5) Continue invoking

Figure 5 The throughput test procedure.

The result of throughput test is shown in Table 2 and Figure 6. Figure 7 shows average of throughput of OpenUCI system based on the different number of workers.

From these results, it shows that OpenUCI system gave a good throughput when the number of workers increased and the increasing of throughput was nearby the increasing of number of workers. For example, the average throughput of 32 workers system was ~6.4 jobs/sec and the average throughput of 1 worker system was ~0.214 jobs/sec. The throughput was increased about 30 times.

5. OpenUCI speed up test

In this test, we observed the run time used to finish jobs when the number of workers was changed from 1, 2, 4, 8, 16, to 32 workers. The procedure of the speed up testing was similar to the throughput testing, but the speed up test changed the number of jobs submitted to system and observed the run time instead of fixing the run time and observed the number of finished jobs.

Table 3 and Figure 8 show the run time of this testing. Table 4 and Figure 9 show the speed up. Table 5 and Figure 10 show the efficiency.

Table 2 The throughtput of OpenUCI.

Time	1 Worker	2 Workers	4 Workers	8 Workers	16 Workers	32 Workers
10	0.20	0.30	0.70	1.40	3.10	6.00
30	0.23	0.43	0.73	1.63	3.17	6.27
60	0.22	0.42	0.82	1.57	3.18	6.33
90	0.21	0.41	0.79	1.64	3.24	6.44
120	0.22	0.43	0.84	1.67	3.31	6.47
180	0.21	0.42	0.83	1.63	3.30	6.59
240	0.21	0.42	0.84	1.67	3.31	6.56
300	0.21	0.42	0.84	1.65	3.31	6.60

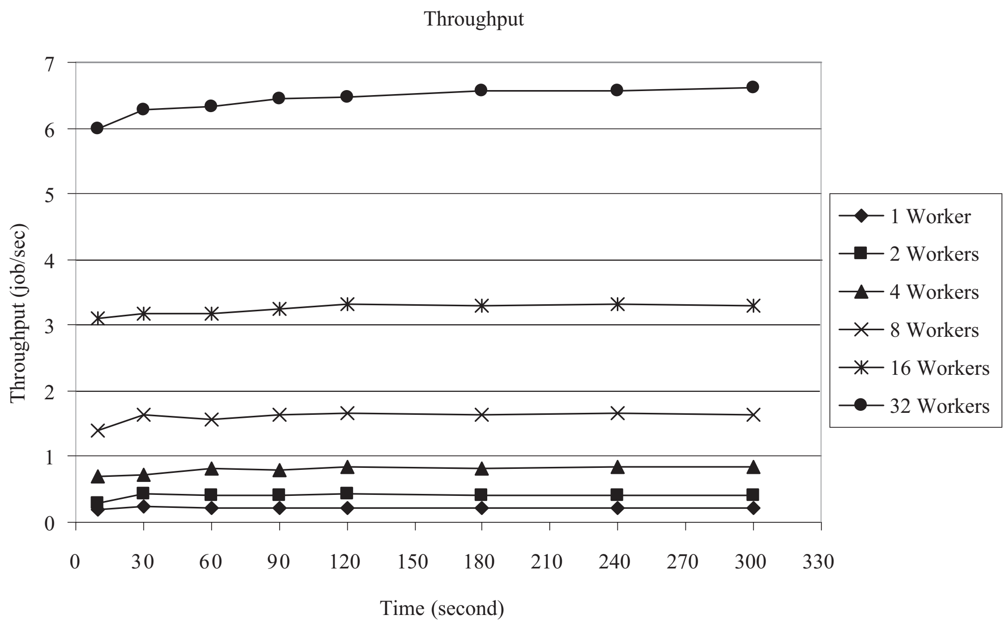

Figure 6 The throughput of OpenUCI system.

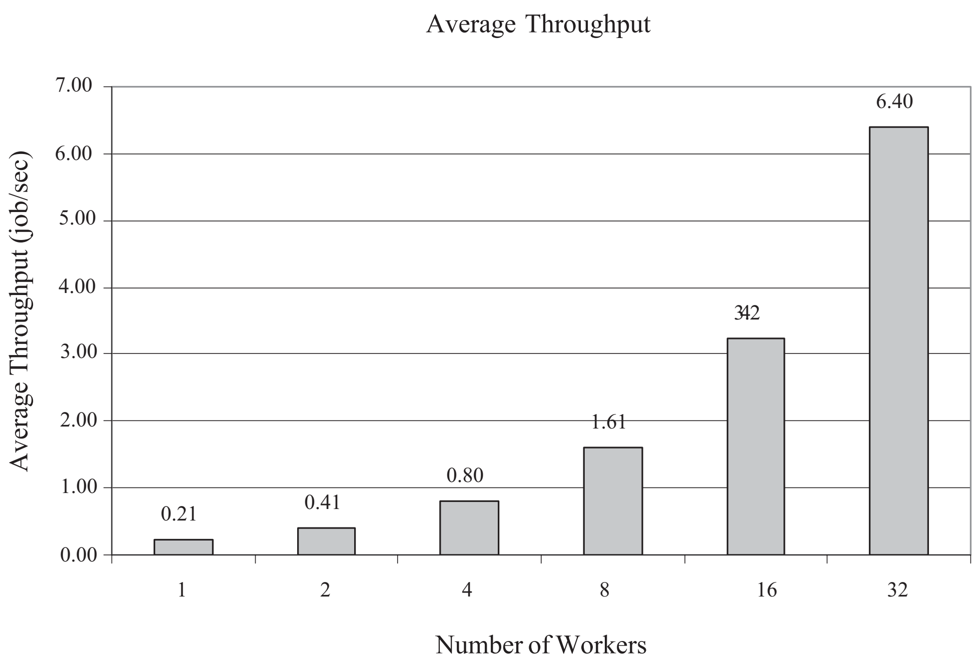

Figure 7 The average throughput of OpenUCI system.

Table 3 The run time of testing (second).

Worker	100 Jobs	500 Jobs	1000 Jobs	2000 Jobs	3000 Jobs
1	476.33	2359.17	4726.77	10083.33	14794.66
2	248.03	1191.59	2400.58	4734.84	7106.32
4	122.14	596.77	1185.05	2386.19	3566.53
8	61.82	303.70	609.31	1216.19	1825.01
16	33.30	157.66	308.28	619.43	923.29
32	19.88	76.25	151.92	301.97	451.55

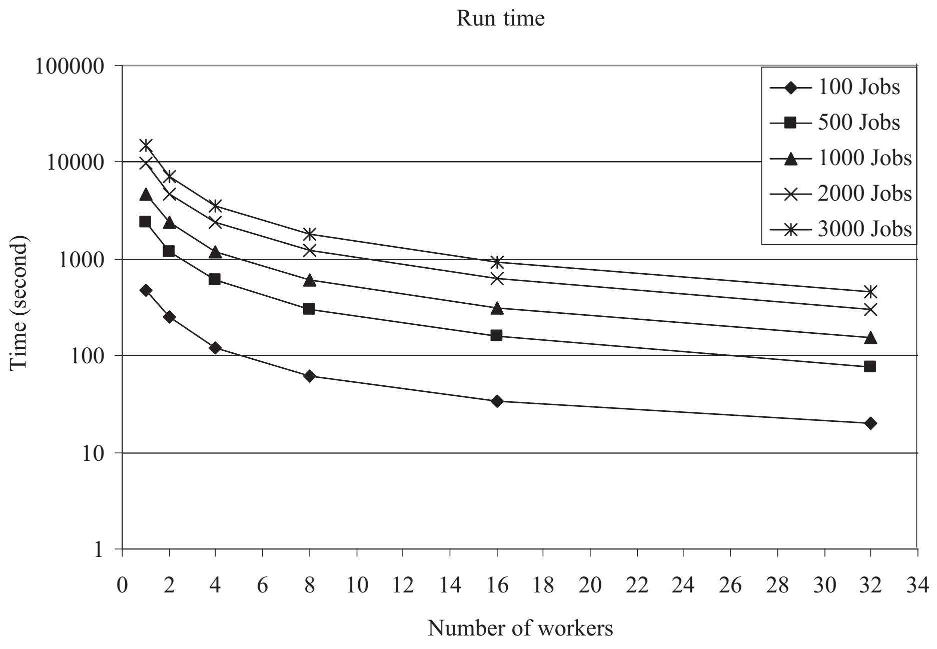

Figure 8 The run time plot.

Table 4 The speed up of testing.

Worker	100 Jobs	500 Jobs	1000 Jobs	2000 Jobs	3000 Jobs
1	1.00	1.00	1.00	1	1
2	1.92	1.98	1.97	2.13	2.08
4	3.70	3.95	3.99	4.23	4.15
8	7.71	7.77	7.76	8.29	8.11
16	14.31	14.96	15.33	16.28	16.02
32	23.97	30.94	31.11	33.39	32.76

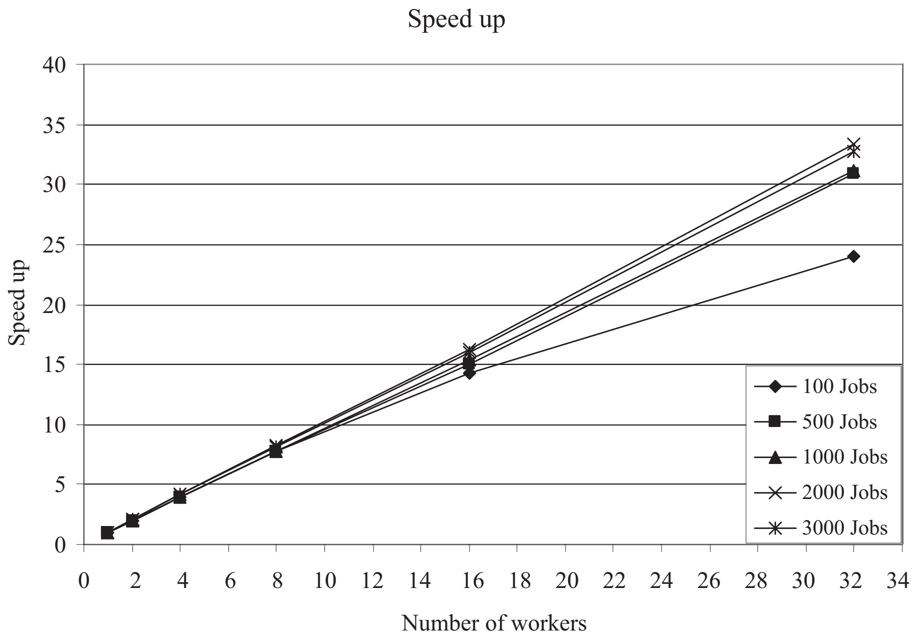

Figure 9 The speed up plot.

Table 5 The efficiency of testing.

Worker	100 Jobs	500 Jobs	1000 Jobs	2000 Jobs	3000 Jobs
1	1.00	1.00	1.00	1.00	1.00
2	0.96	0.99	0.98	1.06	1.04
4	0.97	0.99	0.99	1.06	1.04
8	0.96	0.97	0.97	1.04	1.01
16	0.89	0.94	0.96	1.02	1.00
32	0.75	0.97	0.97	1.04	1.02

The speed up (*S*) of *n*-workers system is defined by the run time of 1-worker system (sequential run time, *Ts*) divided by the run time of *n*-workers system (parallel run time, *Tp*), and the efficiency (*E*) is defined as the speed up (*S*) divided by number of workers (*P*). From Figure 9 and Figure 10, we found that there were three interesting characteristic results.

1. The speed up and efficiency were decreased when the number of workers increased, for example, 100 jobs testing. This characteristic happened because all workers in system are not fully utilized. For example, in 32-workers system, it had to use 4 iterations to finish 100 jobs (32+32+32+4 = 100). So, in the last iteration, there were 28 workers free. Assume that 1 job used 1 second for executeing. The speed up was 25 (Ts/Tp = 100/4 = 25), and the efficiency was 0.78 (S/P = 25/32 = 0.78). If we submitted 128 jobs (32+32+32+32) to this system, the speed up and efficiency would be 32 (128/4) and 1, respectively.

2. The speed up and efficiency were almost perfect. The perfect speed up was the speed up that was equal to number of workers in system. The perfect efficiency was the efficiency that is equal to 1. Basically, the communication overhead

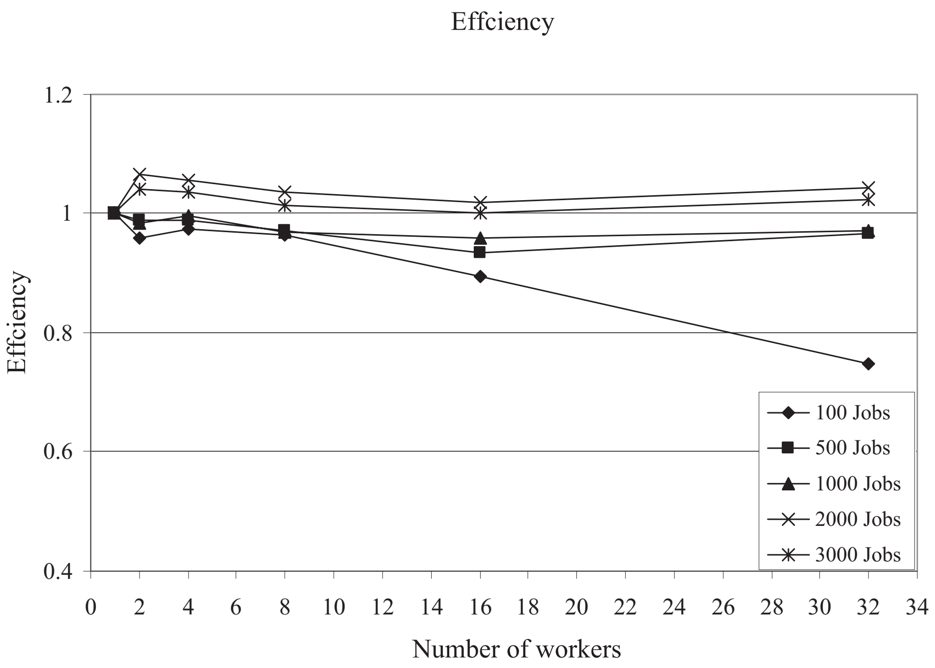

Figure 10 The efficiency plot.

such as input data transfer time makes the speed up and efficiency dropped. In this test, we reduced the data transfer time by replicating VaR database to all workers. So, the efficiency and speed up were nearly perfect.

3. The super speed up and the over efficiency. This characteristic happened because the overhead time before calling web services of client application makes the run time of client application increased. The high number of jobs made the total overhead time grower. However, the total overhead time was reduced by the increasing of number of workers. So, at the large amount of jobs such as 2000 and 3000 jobs, the run times of 2, 4, 8, 16, and 32 workers system were decreased more than the number of workers in system.

CONCLUSION

The demand of using super computing system in organizations has been increasing. They need the system that has more dynamicity and flexibility in order to support the various types and large amount of demand of customers. Moreover, this system must provide an easy and familiar mechanism for customers to use the power of system. This paper proposed the design and implementation of framework used for building the computing environment that can achieve these requirements. This framework is called OpenUCI (Open Utility Computing Infrastructure) which works on Microsoft .NET platform. OpenUCI will gather resources distributed on the network, and automatically adjust and provisioning resources to users. The prototype of OpenUCI has already been implemented and evaluated with a financial engineering application named VaR calculation. The result of evaluation showed that OpenUCI can give a good performance and high utilization when the number of computers and demand of users increased

The prototype version of OpenUCI has only a few modules such as resource collecting and discovery, resource selection and broker mechanism. There are still many necessary modules that should be implemented, for example, web and grid services invoker, job queue manager and virtual computer management. The following is the list of future work. There are many possible works in the future such as integrating the executable file launcher implemented in another related project to OpenUCI system, implementing the job queue management module, implementing the web and grid services invoker module, implementing the virtual computer management service, implementing the user authentication and accounting modules, implementing the data transfer service, exploring the mechanisms for handling fault of machines and jobs and investigating a proper workload distribution scheme and study using simulation.

All these works will make OpenUCI more useful in the modern computing environments.

LITERATURE SITED

Albaugh V. and H. Madduri. 2004. The utility metering service of the Universal Management Infrastructure. **IBM Systems Journal** 43(1): 159-178

Anderson D., J.Cobb, E. Korpela, M. Lebofsky and D. Werthimer. 2002. SETI@home: An Experiment in Public-Resource Computing. **Communications of the ACM** 45(11): 56-61

Eilam T., K. Appleby, J. Breh, G. Breiter, H. Daur, S.A. Fakhouri, G.D.H. Hunt, T. Lu, S.D. Miller, L.B. Mummert, J.A. Pershing and H. Wangner. 2004. Using a utility computing framework to develop utility systems. **IBM System Journal** 43(1): 97-120

Foster I., C. Kesselman, J. Nick and S. Tuecke. 2002. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration**.** Open Grid Service Infrastructure WG. **Globus Grid Forum**

Ganek G. and T. A. Corbi. 2003. The dawning of the autonomic computing era. **IBM System Journal** 42(1): 5-18

Humphrey M. and G. Wasson. 2005. Architectural Foundations of WSRF.NET., **International Journal of Web Services Research** 2(3): 83-97.

Luther A., R Buyya and S. Venugopal. 2005. Alchemi: A .NET-Based Enterprise Grid Computing System. **Proceedings of the 6th International Conference on Internet Computing (ICOMP'05), June 27-30, 2005,** Las Vegas, USA

Rojanapanpat T., P. Uthayopas, S. Chaisiri, J. Pichitlamken, S. Phakhawirotkul and T. Vorakosit. 2005. Implementing a Distributed High Volume Risk Analysis Software on PC Farm using OpenUCI System. **The 9th National Computer Science and Engineering Conference (NCSEC2005), October 27-28, 2005,** Bangkok, Thailand.

Sarmenta L. F. G. 2001. **Volunteer Computing**. Ph.D. thesis, Massachusetts Institute of Technology.

Sarmenta L. F. G., S. J. V Chua, P. Echevarria, J. M. Mendoza, R. R. Santos and S. Tan. 2002. Bayanihan Computing NET: Grid Computing with XML Web Services. **Workshop on Global and Peer-to-Peer Computing at the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid '02), May 2002,** Berlin, Germany.

Smarr L. and C. Catlett. 1992. Metacomputing, pp. 44-52**. Communication of the ACM, 35.**

The Globus Alliance. 2005. Welcome to The Globus Toolkit Homepage. **The Globus Toolkit**. Available source: http://www.globus.org/toolkit/, March 14, 2006.

Thierry P. 2006. CoreGRID: European Research Network on Foundations. **Software Infrastructures and Applications for large scale distributed GRID and Peer-to-Peer Technologies**. Available source: http://www.coregrid.net/, March 14, 2006.