THESIS APPROVAL

GRADUATE SCHOOL, KASETSART UNIVERSITY

Master of Engineering (Computer Engineering)
DEGREE

Computer Engineering	Computer Engineering
FIELD	**DEPARTMENT**

TITLE: Design and Implementation of a Framework for .NET-Based Utility Computing Infrastructure

NAME: Mr. Thanapol Rojanapanpat

THIS THESIS HAS BEEN ACCEPTED BY

_____ **THESIS ADVISOR**
(Assistant Professor Putchong Uthayopas, Ph.D.)

_____ **COMMITTEE MEMBER**
(Associate Professor Surasak Sanguanpong, M.Eng)

_____ **COMMITTEE MEMBER**
(Mr. Pirawat Watanapongse, Ph.D.)

_____ **DEPARTMENT HEAD**
(Mr. Pirawat Watanapongse, Ph.D.)

APPROVED BY THE GRADUATE SCHOOL ON April 5, 2006

_____ **DEAN**
(Associate Professor Vinai Artkongharn, M.A.)

THESIS

DESIGN AND IMPLEMENTATION OF A FRAMEWORK FOR .NET-BASED UTILITY COMPUTING INFRASTRUCTURE

THANAPOL ROJANAPANPAT

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Master of Engineering (Computer Engineering)
Graduate School, Kasetsart University
2006

ISBN 974-16-1401-2

Thanapol Rojanapanpat 2006: Design and Implementation of a Framework for .NET-Based Utility Computing Infrastructure. Master of Engineering (Computer Engineering), Major Field: Computer Engineering, Department of Computer Engineering. Thesis Advisor: Assistant Professor Putchong Uthayopas, Ph.D. 48 pages.
ISBN 974-16-1401-2

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 thesis 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 establish technology such as Grid, Web services, .NET technology to build a virtualized and unify access to resources. Basics 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.

Thanapol R. 28 / 3 / 2006

Student's signature Thesis Advisor's signature

ACKNOWLEDGEMENTS

Thank you, I would like to say “Thank you”. It might be a simple word you hear everywhere, but at this time it is not the same cause it was written from the bottom of my heart.

Thanks to my parents, who take care and support me everything. Even though they have never said anything directly, but their acts always mean “Go ahead, son”. I would like to dedicate this thesis for you.

Thanks to my thesis advisor, Assistant Professor Putchong Uthayopas, Ph.D., and committees, Associate Professor Surasak Sanguanpong, and Mr. Pirawat Watanapongese, Ph.D., who suggest me in everything, not only the classroom knowledge, but the real-world thought as well.

Finally, Thanks to my sisters, Ms. Thanatphat Paneekarn, and Ms. Panida Gunama, who motivate and cheer me up. They always said “Fight Fight”, “Don’t give up”, and “Almost done”. Thanks to my friends and my colleges, Mr. Suphachan Phakhawirotkul, Mr. Theewara Vorakosit, Mr. Ravipol Phutthachartkul, and everyone in High Performance Computing and Networking Center (HPCNC), Faculty of Engineering, Kasetsart University, who give me any suggestions and collaboration.

Thanapol Rojanapanpat
March 2006

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES (Cont'd)

Figure **Page**

DESIGN AND IMPLEMENTATION OF A FRAMEWORK FOR .NET-BASED UTILITY COMPUTING INFRASTRUCTURE

INTRODUCTION

Problems

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. First they can outsource the computing power. The other solution is to create the supercomputing system by using the already existing personal computers (PC) in their company. The creation of a supercomputing system from normal computers or desktop PCs now is not an imagination, because the speed and performance of personal computers has been increasing. Furthermore, the speed and bandwidth of network also increase too. From these two advantages, they cause emerging of new computing systems. One of them is the utility computing system.

Utility computing (Eilam et al., 2004) is the next computing model that involves the use of many diverge technology such as grid computing (Foster et al., 1998, 2001, 2002), autonomic computing (Ganek and Corbi, 2003). Utility computing system focuses on the creating of virtual computing environment which dynamically and automatically virtualizes, provisions, and manages resources and services on demand of users. The major benefits of utility computing are;

1. More utilization – the resources in utility computing system can be shared and used in the efficient way,

2. More flexibility – utility computing system provides flexibility in the creation of the dynamic computing environment which can automatically increase and decrease the computing resources corresponding to the demand of users, and

3. Lower total cost of ownership – utility computing can provide IT and business process outsourcing which can reduce cost of investing in resources such as hardware assets, maintenance cost, training cost, etc.

However, the designing and building of utility computing infrastructure is still difficult and challenge task. We classify the problem into two groups.

1. Resources virtualization – Due to the resources that we use to build utility computing infrastructure are normal personal computers that have a high dynamism, the resources can be available and unavailable on the 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.

2. Resources provisioning – the 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 the demand of users increased. After that, 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.

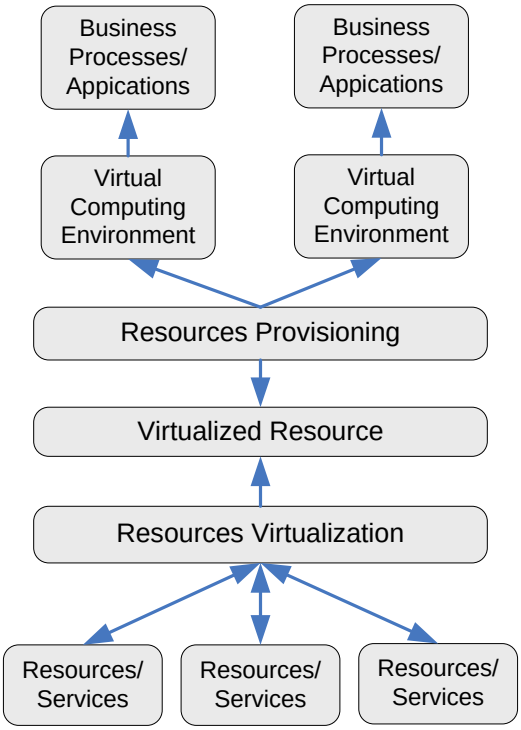

Figure 1 The relationship of resource virtualization and resource provisioning

In this thesis, we present a design and implementation of a framework called OpenUCI (Open Utility Computing Infrastructure) which is used to construct 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 to implement the OpenUCI system, because it provides a powerful and easy 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. However, we need the OpenUCI framework to be used both Web service's world and Grid service's world. So, we choose the WSRF.NET (Web Service Resource Framework .NET) (Humphrey and Wasson, 2005), which is a tool for implementing Grid service on .NET platform.

Objectives

1. Design and build a utility computing infrastructure using Microsoft Windows based on PC farm.

2. Explore the issue of how to merge Grid technology with Microsoft's .NET technology.

3. Engage in solving related research problems such as

   - The resource discovery technique
   - The resource allocation on demand technique

Research Scope

1. Propose architecture for creating the utility computing infrastructure.

2. Design and implement a prototype utility computing system based on Microsoft's .NET technology.

   - Evaluate the proposed system.

LITERATURE REVIEW

Background Knowledge

Utility computing system must be based on current distributed system technology. In (Thierry, 2006), a good survey of platform technology that is available is provided. 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 following.

Grid computing (Foster et al., 1998, 2001, 2002) focuses on integrating geographically distributed resources into a unified system. Grid computing provides concept of Virtual Organization (VO) which is a 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 sharing processing power rather than a sharing of files. The traditional P2P has three generations classified by the P2P topology.

1. First generation (Centralized model) – The topology of the first generation had a centralized server that keeps some information of all peers such as list of shared files. The Advantage of this model is it has less complexity, because the discovering shared resources only occurs in the central server, but the disadvantage is the model cannot serve the system that has a lot of client peers. The popular project is Napster (for more information at http://www.napster.com)

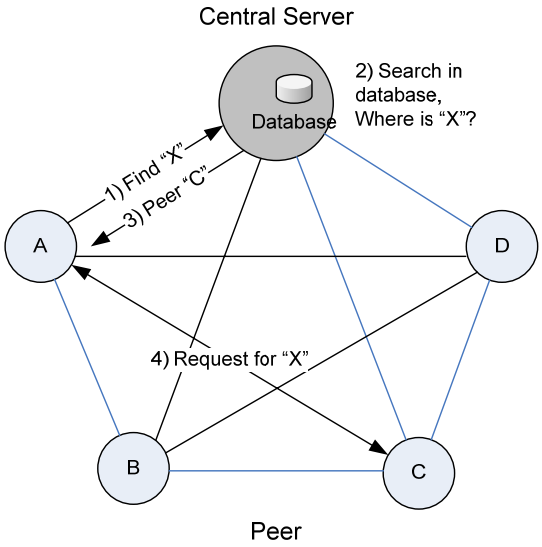

Figure 2 The first generation of P2P model

2. Second generation (De-centralized model) – The second generation is also called pure P2P or de-centralized system, in which there is no need of a central server. Every peer in this model has the same capability, and they can share resources directly. The advantage of this model is it has more robustness and fault tolerant. The example software is GNUtella (for more information at http://www.gnutella.com)

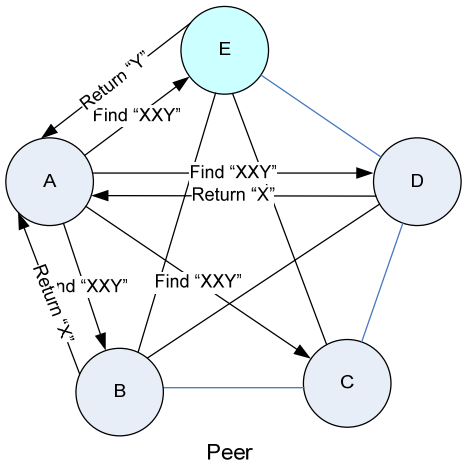

Figure 3 The second generation of P2P model

3. Third generation (Hybrid model) – The third generation is an improvement of the first and second generations. It has a "super-peer", the peer that has more capability such as faster CPU, higher network bandwidth, etc. The super-peer will connect to each others to perform a backbone of system, and it is responsible for managing, or controlling its client peers. The advantages of the model are it is easy to manage and control bandwidth of client-peers, and it reduces a searching time. The example software is KaZaA (for more information at http://www.kazaa.com)

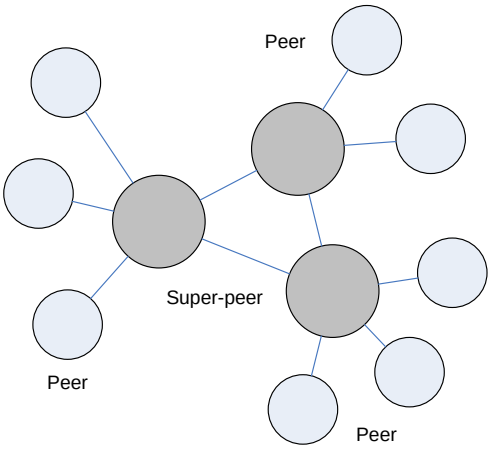

Figure 4 The third generation of P2P model

Volunteer computing (Sarmenta, 2001) focuses on making computers to be a part of metacomputer dynamically when computing power are 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 dedicate itself to execute jobs. This includes system such as SETI@home (Anderson et al., 2002), Bayanihan (Sarmenta et al., 2002), and Alchemi (Luther, 2005).

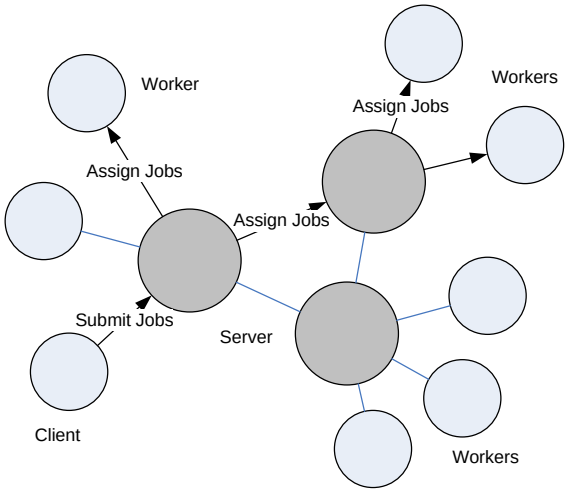

Figure 5 The volunteer computing model

Utility computing system usually consists of resources that can be any platforms such as Windows, Linux, and etc. Perhaps, it has to communicate with other systems or applications. To eliminate the platform dependency, and make utility computing system can seamlessly work with any other systems or applications. The SOA (Service Oriented Architecture) technology is used to develop the OpenUCI framework.

There are three basic components of SOA, service provider, service requestor, and service registry. The service providers provide useful services to service requestors.

1. Service providers will publish their service descriptions to the service registries.

2. Service requester is the users that search for desired services from the service registries. Once, the resource requester receives a matched service description from the service registry, it requests for accessing the service on the service provider.

3. Service registry is responsible for advertising service descriptions published by the service providers, and finding service descriptions needed by the service requesters.

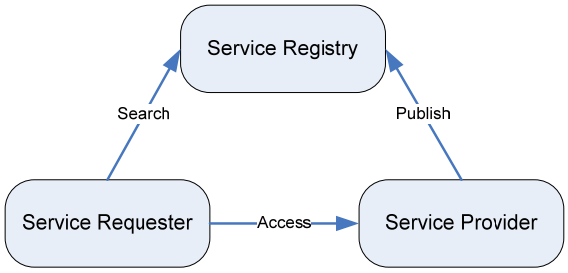

Figure 6 The basic components of the Service Oriented Architecture (SOA)

The well-known implementations of SOA are web services and grid services technologies. There are some different between web and grid services that we will present in the next. In the OpenUCI project, we use .NET web services to implement the core services of OpenUCI framework such as resource collecting, resource discovering, and job submission. However, we want to explore the solution for merging web and grid services, which WSRF.NET (Web Service Resource Framework .NET) is responsible for this task.

Web services technology is a collection of protocols and standards used for interoperability between applications. The web services can be used for exchanging data, remote execution of different platform applications, and they are platform-independence and language-independence because they use standard XML language in communication between applications. The figure 7 is the general web services architecture.

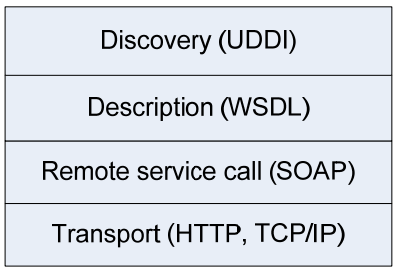

Figure 7 The web services architecture

In the web services protocol stack, there are three interesting terms, 1) UDDI, Universal Description, Discovery and Integration, is a catalog of web services. UDDI is used to register and discovery web services, typically described in WSDL, 2)

WSDL, Web Services Description Language, is used to describe the public interface to web services in which how to communicate, interact, and invoke web services, 3) SOAP, Simple Object Access Protocol, is a message-based protocol based on XML. The SOAP specification defines a mechanism for exchanging of information, and invoking the operations of web services. The relationship of UDDI, WSDL, and SOAP can be shown by this diagram.

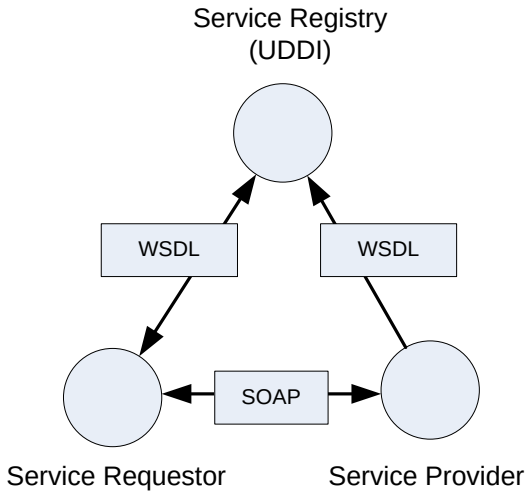

Figure 8 The relationship of UDDI, WSDL, and SOAP

Grid services technology – Open Grid Services Architecture (OGSA) (The Globus Alliance, 2005b; Foster et al., 2002; Unger and Haynos, 2005), the architecture and specifications of grid services are defined by the Open Grid Services Infrastructure (OGSI) working group of the Global Grid Forum (GGF). Grid services enhanced the web services functionality to address grid requirements. Grid service was extended the functionalities for discovery, life cycle, state management, creation and destruction, event notification, and reference management.

WSRF.NET is an implementation of the full set of specifications for WS-ResourceFramework (WSRF) and WS-Notification (WSN) on the Microsoft .NET framework (Humphrey and Wasson, 2005). WSRF.NET is a set of software, libraries, tools and applications used as plug-in for Visual Studio .NET in order to enable the developers can develop grid services, which are already based on WSRF and WSN specifications, on top of .NET framework. There are four specifications defined by WSRF (Snelling et al., 2006).

1. WS-Resource Properties defines how data associated with a stateful resource can be used by web services.

2. WS-Resource Lifetime defines how WS-Resource can be destroyed.

3. WS-BaseFault defines an XML schema type for error message types of web services.

4. WS-ServiceGroup defines how WS-Resource having the same propose can be grouped together in order to be searched and queried easily

The WS-Notification is not a part of WSRF. The WSN have three normal specifications (Vambeneqe et al., 2006).

1. WS-BaseNotification defines the Web services interfaces for NotificationProducers and NotificationConsumers. It includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with operational requirements expected of them.

2. WS-BrokeredNotification defines the Web services interface for the NotificationBroker. A NotificationBroker is an intermediary which, among other things, allows publication of messages from entities that are not themselves service providers.

3. WS-Topic defines a mechanism to organize and categorize items of interest for subscription known as “topics.”

Related Frameworks

Globus (The Globus Allience, 2005) is a middleware the enable your machines to connect to the grid. Globus provides APIs, and utilities to access, and manage grid resources. The architecture of Globus consists of one core component, security, and three core pillars built on top of the security component. The three pillars are 1) Execution management deals with job management, job scheduling, and program execution, 2) Information service is responsible for monitoring, collecting and discovering grid resources and system information, and 3) Data management handles the accessing, transferring data between grid resources.

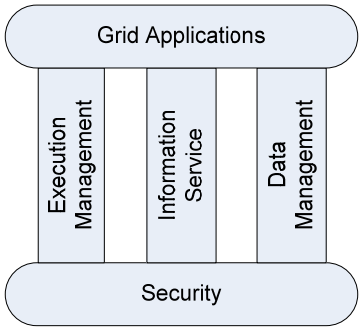

Figure 9 Globus key components

GUSTOFarm (Chaisiri and Uthayopas, 2005) is a utility computing framework based on peer-to-peer approach. The architecture of GUSTOFarm is similar to OpenUCI, but it differs in implementation technology and topology of system. GUSTOFarm is implemented with Java technology, but OpenUCI uses .NET technology. The topology of GUSTOFarm is fully decentralized P2P model, but OpenUCI is designed to hybrid P2P model.

SETI@home (Anderson et al., 2002) – The Search for Extraterrestrial Intelligence is based on volunteer computing. It uses a million of idle computers in homes or offices around the world to analyze radio signal from outside Earth. SETI@home is also classified to centralized P2P model. SETI@home did not provide ability to submit jobs, discovery resources, and etc. need by a utility computing framework. The SETI@home client program only requests a job from server, executes it, and then send result back. However, the mechanism of using idle computers is applied to the OpenUCI framework.

Alchemi (Luther et al., 2005) is a .NET-based grid computing framework that provides the runtime machinery and programming environment required to construct desktop grids and develop grid applications. The topology of Alchemi can be classified into the hybrid (hieratical) P2P model. Alchemi is implemented by Microsoft's .NET technology. Alchemi uses .NET Remoting for interaction between its components, and it also provides web service for communicating with other grid systems. However, Alchemi lacks of resources discovery, data management service, and native programs execution (e.g. Windows' exe program). Alchemi can execute jobs that are implemented with its programming paradigm.

Table 1 The comparison of OpenUCI and some related systems.

	Globus (GT.4)	**GUSTO Farm**	**SETI@ home**	**Alchemi**	**OpenUCI**
Topology	Hybrid	Decentralized	Centralized	Hybrid (hierarchical)	Hybrid
Implementing technologies	Java, C, Grid Services.	Java, Web services	C++	C#, .NET technology	C#, .NET technology, WSRF.NET
Broker Mechanism	Yes	Yes	No	Yes	Yes
Execution Management	Yes	Yes	Partial[1]	Partial[2]	Yes
Resource Management	Yes	Yes	No	No	Yes
Data management	Yes	N/A	N/A	No	Yes
Automatic adjustable computing environment	No	No	No	No	Yes

[1] It can execute only a specific program. Users submit job to system.

[2] It supports only programs developed by using its' paradigm. It does not support the execution of native programs (.exe file) or web/grid services software

Related Commercial Products

Many related ideas have been proposed by companies as a future trend for the next generation IT. For example, the On-demand Business Model (Albaugh and Madduri, 2004; Eilam et al., 2004) by IBM, which proposes the universal management infrastructure (UMI), used to build an environment for demand services. This includes a Flexible Hosting services, IBM Workplace Collaboration services.

HP Utility Data Center (UDC) (HP, 2006) is a set of products for data center deployment. The HP UDC consists of *HP BladeSystem*, management tool for configuring and dynamically allocating virtual resources. *HP Virtual Server Environment*, a virtual infrastructure that can automatically grows and shrinks resources in real time. *HP OpenView Change and Configuration management solutions*, tools for automatically changing and configuring resources in order to reduce operational costs. HP Utility Services: HP Managed services and HP Financial services that use the concept Pay-As-Use.

Sun Grid and Managed Services (Sun Microsystems, 2006) are solutions building on the utility computing concept. For Sun Grid, it offers computing power and storage which you can purchase it as you need, without long-term cost. The price of Sun Grid Compute Utility is $1/cpu-hr, and $1/GB-mo for Sun Grid Storage Utility. For Sun Managed Services, it provides IT management services to help you enhance the business value.

For Microsoft, a framework called Microsoft Dynamic Systems Initiative (DSI) (Microsoft, 2006) has been proposed. Microsoft DSI has three key components. First, Cross-product roadmap, a set of product such as development tools, operating system, server applications, and management tools. Second, The System Definition Model (SDM), SDM is a model used to create definitions of distributed systems. It defines a set of related software and hardware resources working together. Finally, Board partner support, a third party software and hardware support component.

MATERIALS AND METHODS

Materials

Hardware requirement

This thesis has to use Windows-based personal computers to develop and test a framework. The computers used in this development comprise of 1 node manager, 32 nodes workers, and 1 node user. All nodes are connected using Fast Ethernet switch. The topology and specification of computers are shown in Figure 10 , and Table 2 , respectively.

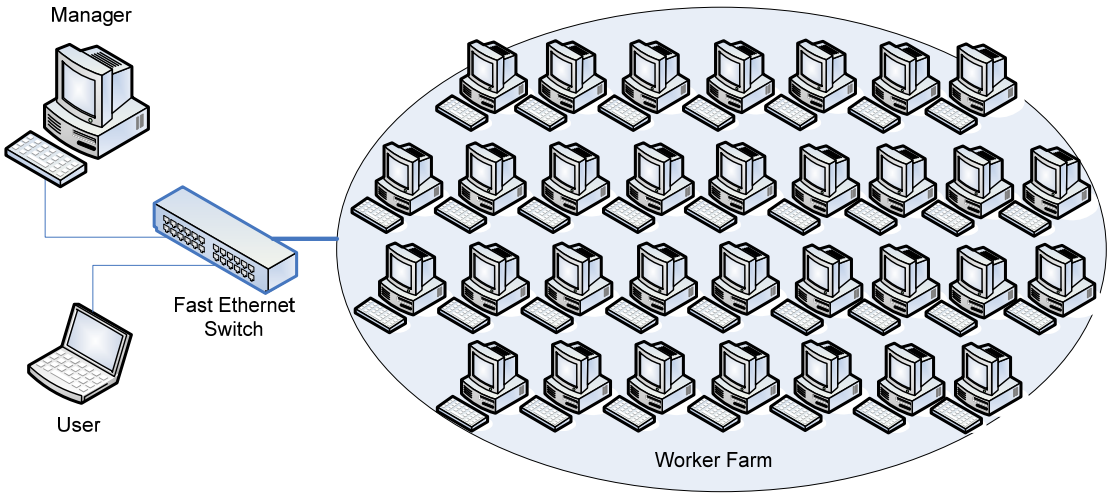

Figure 10 The Windows cluster

Table 2 The hardware specifications

Machines	Hardware	Operating system
Manager	AMD Athlon 2.0GHz, 512 MB RAM	Windows server 2003
Workers	Intel Celeron 2.53GHz, 512 MB RAM	Windows XP Professional
User	Intel Pentium M 1.5GHz, 768 MB RAM	Windows server 2003

Software requirement

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

Method

The development of UCI uses the iterative model. We will develop the necessary HIC components with simple functionalities in order to confirm that it is stable, and then we extend more functionality and add some new algorithms. The methodologies of this research are as follow;

1. Analyze the outstanding functions of other frameworks such as Alchemi, Bayanihan, GUSTOFarm, BONIC, etc.

2. Design the UCI architecture, components, basis services, and basis API.

3. Implement a prototype version of each component.

4. Test the system.

5. Design and implement more features.

6. Evaluate and tune performance of system.

7. Publish a paper and a thesis.

Framework Design and Implementation

This chapter shows the final version of the OpenUCI framework. The architecture, components, and necessary state diagram and API of the OpenUCI framework will be shown in the next subsections. After that, we will describe the implementation of the prototyped of OpenUCI framework.

Requirements

In this thesis, the utility service is the functions provided by any computers. The utility service must base 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. The resources can be computing power (CPU), storage, files, and utility services.

According to the utility computing system development problems that we state in the introduction section, *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* used to gather 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, etc.

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

Components

There are three main components in OpenUCI system.

1. *Manager* is the computers that provide core services used to manage shared resources, and support incoming requests of users

2. *Worker* is the 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. The 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. *User* is 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.

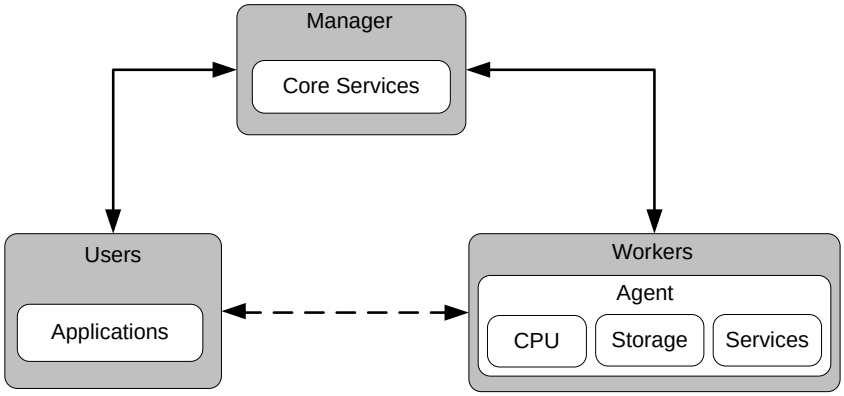

Figure 11 The interaction of manager, worker and user

Architecture

The architecture of the OpenUCI framework is shown in Figure 12 . There are four layers of the OpenUCI framework, resources, .NET platform, core services, and applications.

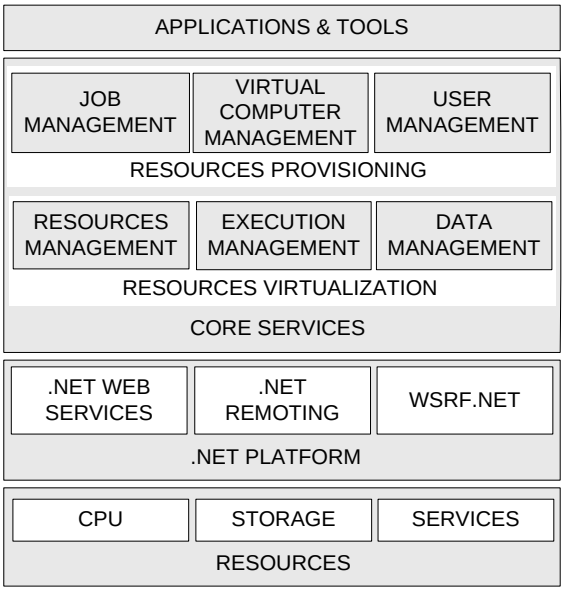

Figure 12 The OpenUCI architecture

Resources layer

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

.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.

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 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 services 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 with the authentication, authorization, accounting, and profiling of users

2. Virtual Computer Management service (VCMS) is used to manage and control the virtual computing environment created by users.

3. Job Management service (JMS) is used to create jobs and support job submission from users. JMS also provides job queuing and scheduling mechanisms.

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.

Detail Design

In this section, we will describe the detail of OpenUCI core services layers. The state diagram, API list, and database schema of necessary core services will be described.

Resources Management Service (RMS)

Components and state diagram

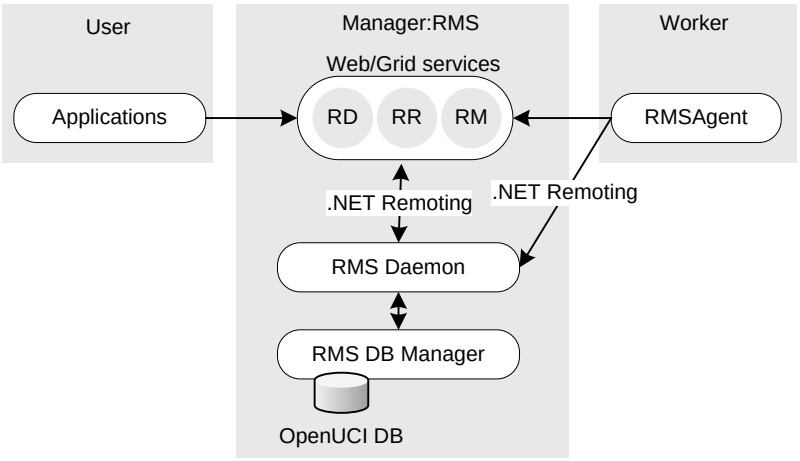

Figure 13 The components of Resources Management Services and its interactions

The resource management service (RMS) provides three important sub-services, Resource Discovery (RD), Resource Reservation (RR), and Resource Monitoring (RM).

1. Resource Discovery service (RC) is responsible for collecting resources information sent by the Resources Management Service Agent (RMSAgent) that is installed in all worker machines. The RMSAgent is a small program used to retrieve necessary information about workers and send it to the manager. The Resource Discovery service also provides mechanism for checking the status of all workers.

Figure 14 shows the state of collecting resources on the manager machine. After the resource collecting module was started, it created two threads. The first thread will wait for registration messages sent by the RMSAgent run on worker machines. When it receives a registration message, it will check the message type. For example, if message type is “Register”, it will add resource description, which comes along with registration message, to the database, and it then increases the time-to-live (TTL) of that worker. The second thread is used to checking the status of all workers by periodically sending “Heartbeat” message to workers. After it sent heartbeat to a worker, it will decrease TTL of the worker.

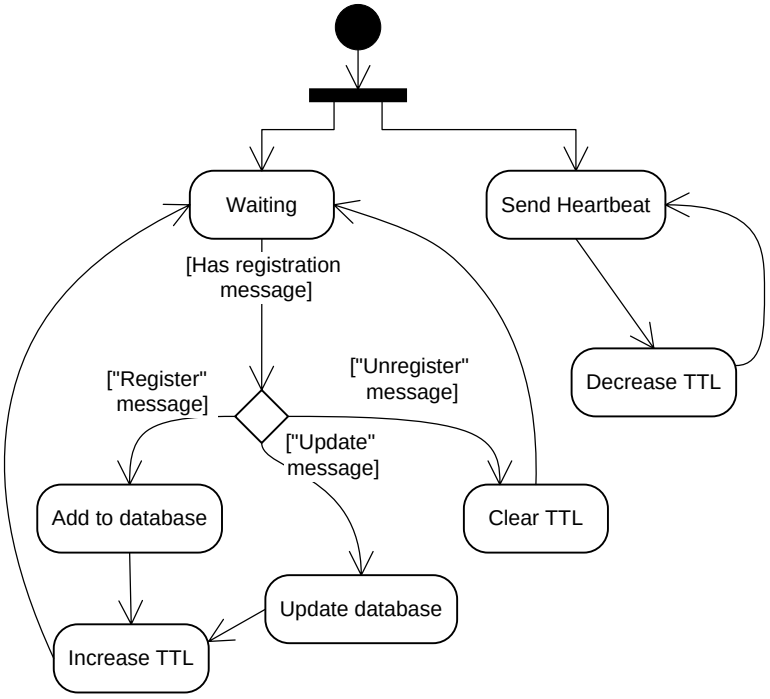

Figure 14 The state diagram of Resources Collecting service

Figure 15 shows the state of the resources management service agent (RMSAgent) program run on worker machines. RMSAgent has two operation modes depended on worker types, dedicated and non-dedicated workers. In non-dedicated mode, RMSAgent will start in idle mode, and wait until worker's CPU is not busy, and it then gets resource information and requests for job form the manager. In this mode, RMSAgent will not send register, update and unregister messages. It only requests or executed jobs when its' CPU is idle, and stop executing jobs when its' CPU is busy. In dedicated mode, RMSAgent has to send resgister, update, and unregister messages to manager. RMSAgent has two ways to update resources information, active update, and passive update. For active update mode, RMSAgent will send update message to manager in some interval time. For passive update, RMSAgent will wait for "Heartbeat" message from manager, ant it then send update message back.

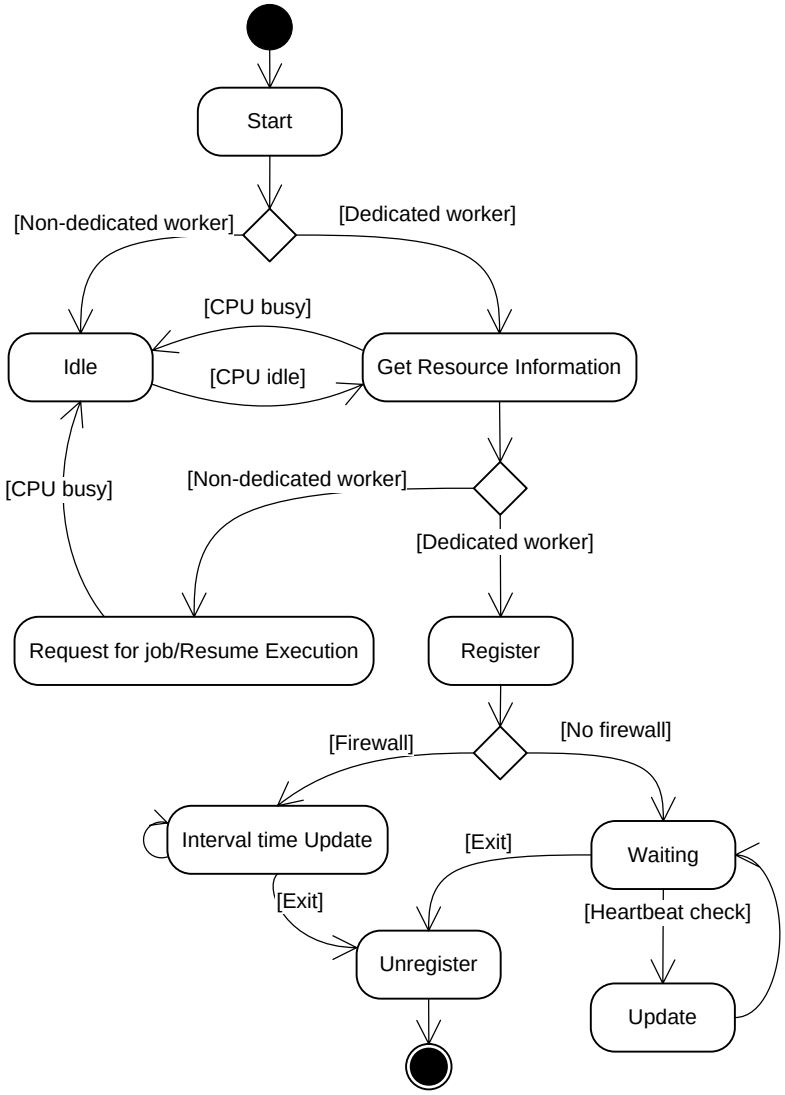

Figure 15 The state diagram of Resources Management Service Agent

The following code is the example of worker's resource description which RMSAgent sent to RMS on manager machine.

```
<Resources>
   <Worker id="FTci4Y/8kkDMS30cEVbeaA">
      <SystemInfo>
         <Hostname name="worker1"/>
         <Network ip="158.108.34.44"
            mac="00:0C:6E:3B:95:50"
            speed="10000"/>
         <Processor num="1"
            speed="2004" busy="54"/>
         <Memory size="512"
            used="386" free="126"/>
         <SharedStorage size="10"
            used="0" free="10"/>
         <OperatingSystem
            name="Microsoft Windows XP Professional/>
         <Statistic loadAvg="34"/>
```

```
         <Performance flops="151"/>
      </SystemInfo>
      <ServicesInfo>
         <Service id="depmKfye5nT2dbGTJmHQ0g"
            name="VaRService"
            wsdl="158.108.34.4/4VaService.asmx?wsdl"
            description="Value at Risk Calculation"/>
         <Service id="0kRBXkr2KbqyKhaIJOxIzQ"
            name="RenderService"
            wsdl="158.108.34.44/RendeService.asmx?wsdl"
            description="3D Graphic rendering"/>
      </ServiceInfo>
   </Worker>
</Resources>
```

In additional, Resource Discovery service is used for user applications or other OpenUCI core services to search desired resources; in this case we call them as requestor. The requestor construct the query string and send to RD service, and then the RD service looks up in the database, and returns the best suitable resources to the requestor. The pattern of query string should look like this.

```
[Resource Type=Value] & [Resource Type=Value] | [Resource
Type=Value]...
```

Figure 16 The pattern of query string for discovery resources.

The resource selection in OpenUCI is performed when there are more than one result (resource) returned by resource discovery process. The resource selection uses Resource Rank metrics to indicate that which resources is the best. The resource rank is calculated from the following formulas.

$$Rank = CpuLoadAvg * w_1 + MemLoadAvg * (1 - w_1)$$

$$CpuLoadAvg = \left(\sum_{i=1}^{n} CpuLoad_i \right) / n$$

$$MemLoadAvg = \left(\sum_{i=1}^{n} MemLoad_i \right) / n$$

$$CpuLoad = 100 - \%CpuIdle$$

$$MemLoad = (MemUsed / MemSize) * 100$$

Figure 17 The resource rank calculation

From Figure 17 , Resource Rank (Rank) depends on two variables, CPU usage average (*CpuLoadAvg*), and memory usage average (*MemLoadAvg*). The lower resource rank is the better resource. The *CpuLoadAvg* and *MemLoadAvg* are calculated from the average of the last *n* time of *CpuLoad* and *MemLoad*, respectively. The *CpuLoad* is the inverse of CPU idle getting from the performance counter of OS. The *MemLoad* is the percent of the memory usage.

2. Resource Reservation service (RR) provides functions to users applications for reserving and release the resources.

3. Resource Monitoring service (RM) provides functions to users applications for query the status and information of resources and system.

The API List

Table 3 The Resources Management Service API list

Functions	**Description**
Resources Collecting	
`bool Register (WorkerInfo resourceInfo)`	`Register worker`
`bool Update (ResourceInfo resourceInfo)`	`Update worker information`
`bool Unregister (string resourceID)`	`Unregister worker`
Resources Discovery	
`ResourceInfo[] Discover (string queryString)`	`Discovery resources using query string pattern`
`ResourceInfo DiscoverTheBestWorker()`	`Discovery the best worker`
`ResourceInfo[] DiscoverByWorkerName (string wName)`	`Discovery workers by worker name`
`ServiceInfo[] DiscoverByServiceName (string sName)`	`Discovery web services by service name`
Resource Reservation	
`bool ReserveWorkers (string[] workerID, long startTime, long endTime)`	`Reserve workers`
`bool ReleaseWorkers (string[] workerID)`	`Release workers`
`Resource Monitoring`	
`string GetManagerStatus()`	`Get the status of manager`
`string GetWorkerStatus (string workerID)`	`Get the status of worker`

Database Table

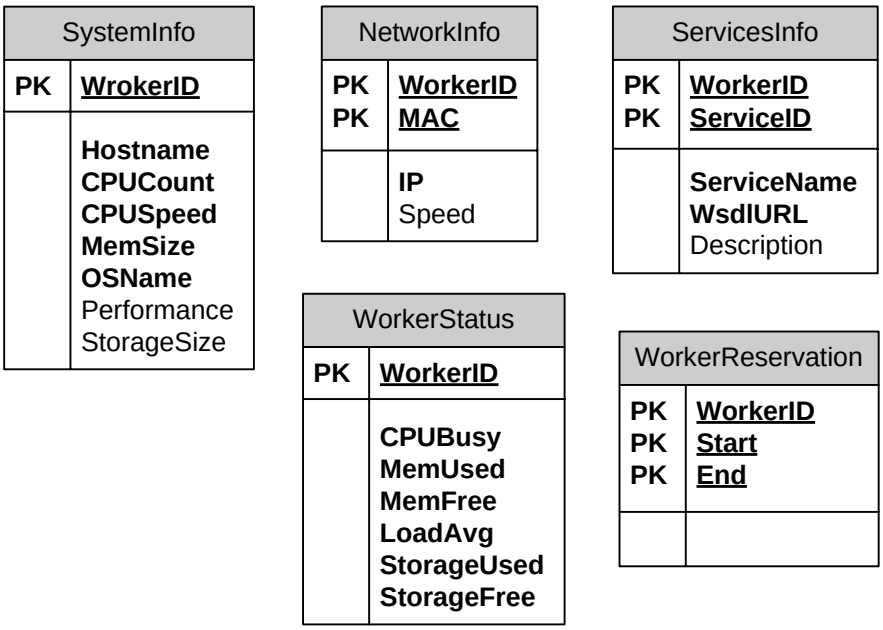

Figure 18 The database table of Resources Management Service

Execution Management Service (EMS)

Components and state diagram

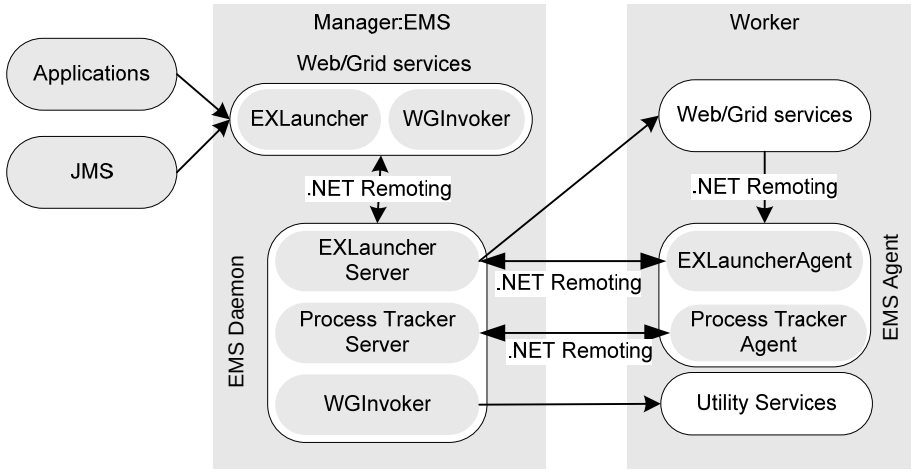

Figure 19 The components of Execution Management Service and its interactions

The execution management service has two sub-services, the executable file launcher service (EXLauncher), and web and grid services invoker service (WGInvoker).

1. Executable file Launcher service (EXLauncher) is used to start, stop, resume, and kill processes running on worker machines.

2. Web and Grid services Invoker service (WGInvoker) is used to invoke the utility services shared on worker machines

In order to track the status of jobs that were launched or invoked on worker machines, ProcessTracker server and agent will be responsible for doing this task.

Figure 20 shows the state of the execution management service daemon. When the EMSDaemon was started, it will create two threads. The first thread is used to wait for incoming jobs submitted by users or JMS. After that, the EMS will check the type of job. If its' type is to launch an executable file, it will dispatch the job to the EXLaucher server, and it then add the job to executable file tracking list (ExeTracking list) of the process tracker server. If the type of job is to invoke the utility service, the EMS will dispatch the job to WGInvoker, and it then add the job to service tracking list (ServiceTracking list) of the process tracker server. The second thread is the process tracker server thread. The state of process tracker server thread will be shown and described in Figure 22 .

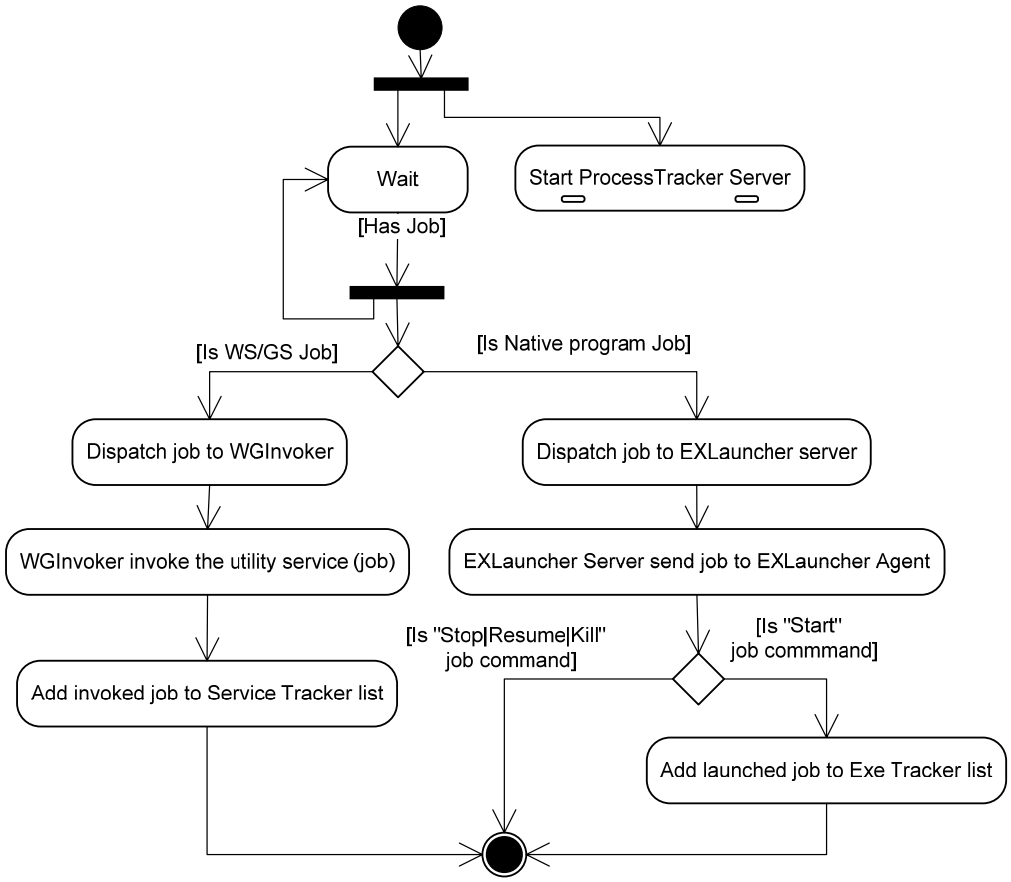

Figure 20 The sate diagram of Execution Management Service Daemon

Figure 21 shows the state of the execution management service agent (EMSAgent). After the EMSAgent was started, it will create the EXLauncher agent, and ProcessTracker agent threads. Once the EXLauncher agent receives job from EXLaucher server, it will check the job command. If job command is to start new process, it will start a process, and then add job to the process tracking list.

ProcessTracker agent uses this list to monitor the status of started processes. For the state and description of ProcessTracker agent will be shown in Figure 23 .

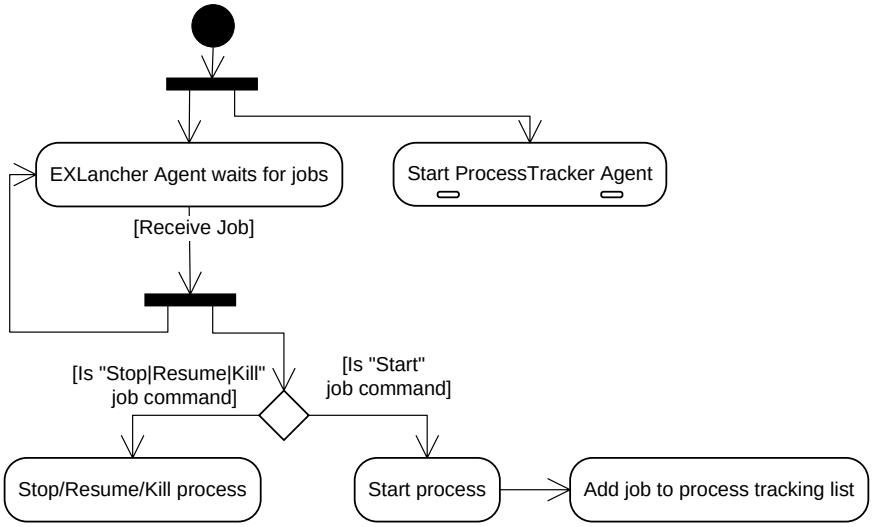

Figure 21 The state diagram of Execution Management Service Agent

Figure 22 and Figure 23 show the state of ProcessTracker server and agent, respectively. The purpose of ProcessTracker is to check the status of jobs both executable file jobs and utility service jobs. There are two ways to check the status of jobs, ServiceTracking and ExeTtracking. The ServiceTracking uses the web or grid services invocation handles returned by WGInvoker to poll the status of invoked utility services. The ExeTracking is used to check the status of executable file jobs. The ExeTracking will wait for the “Job Finished” message from the ProcessTracker agent.

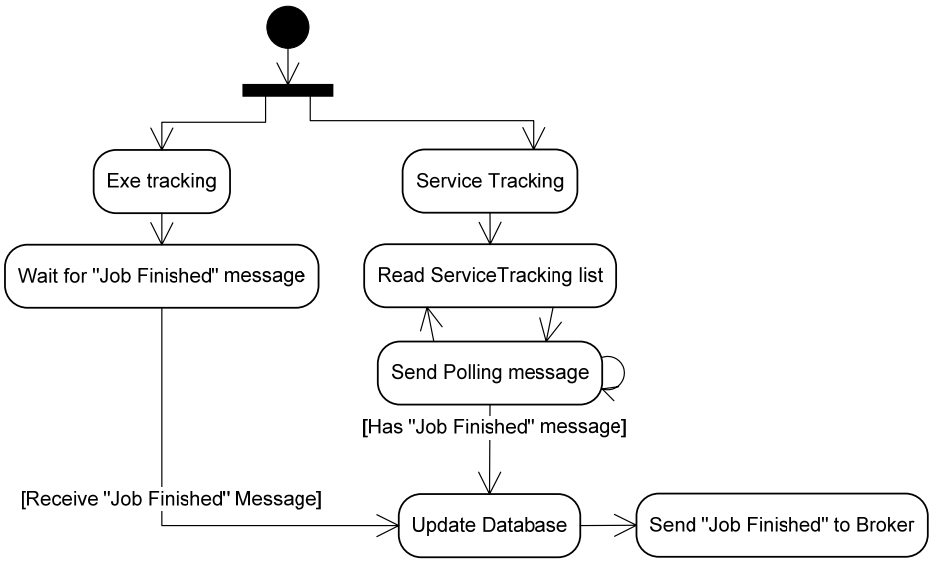

Figure 22 The state diagram of ProcessTracker Server

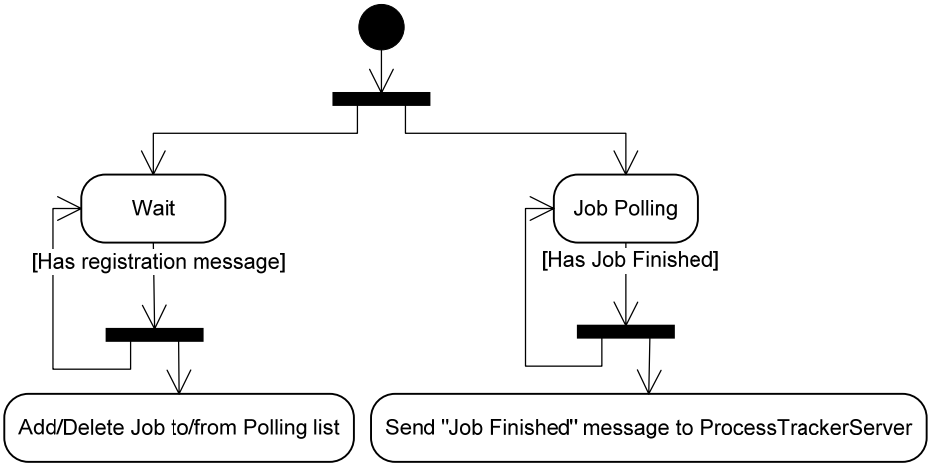

Figure 23 The state diagram of ProcessTrackerAgent

The API List

Table 4 The Execution Management Service API list

Functions	Description
`int EXStart (string workerIP, string stdin, string stdout, string exeFile, string[] args)`	`Start executable program job`
`int WSStart (object result, string wsdlURL, string[] args)`	`Start web service job`
`int GSStart (object result, string wsdlURL, string[] args)`	`Start grid service job`
`bool Stop (int jobID)`	`Stop job`
`bool Resume (int jobID)`	`Resume job`
`bool Kill (int jobID)`	`Kill job`

Data Management Service (DMS)

Components and state diagram

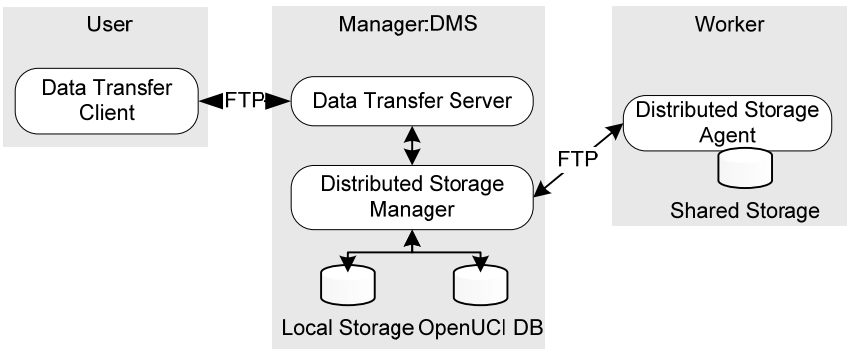

Figure 24 The components of Data Management Service and its interactions

The Data Management Service uses FTP protocol to transfer files among user, manager, and worker. The Distributed Storage Manager is used to decide that the files uploaded by users will be save on local storage or should be distributed to shared storage on worker machines.

The API list

Table 5 The Data Management Service API list

Functions	Description
`bool DMSLogin (string url, int port, string username, string password)`	`Login to DMS Server`
`bool DMSLogout ()`	`Logout from DMS Server`
`string DMSPut (string localFile)`	`Upload file to server`
`bool DMSGet (string remoteFile, string localFile)`	`Download file from server`
`bool DMSDel (string remoteFile)`	`Delete file on server`
`string[] DMSList ()`	`List file on server`
`bool DMSCD (string directory)`	`Change directory`

Database Table

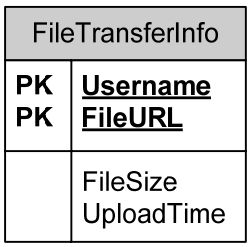

Figure 25 The database table of Data Management Service

User Management Service (UMS)

Components and state diagram

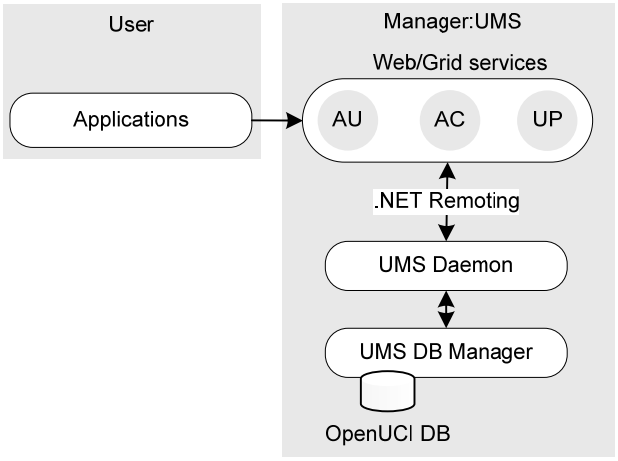

Figure 26 The components of User Management Service and its interactions

There are three basic sub-services in the User Management Service.

1. Authentication service (AU) provides APIs for users to login and logout from the OpenUCI system.

2. Accounting service (AC) is used to collect the resources usage of users, and calculate the usage cost.

3. UserProfiling and Permission service (UP) is used to set or get the basic descriptions and permissions of users.

The API list

Table 6 The User Management Service API list

Functions	Description
Authentication	
UCIManager UCILogin (string url, int port, string username, string password)	Login to manager
bool UCILogout ()	Logout from manager
Accouting	
int GetCost (string costType)	Get total cost
int GetUsage (string usageType)	Get total usage
int GetUsageRate (string rateType)	Get usage cost rate
bool SetUsageRate (string rateType, int value)	Set usage cost rate
int GetQuota (string quotaType)	Get quota of usage
bool SetQuota (string quotaType, int value)	Set quota of usage
UserProfiling and Permission	
string GetFullName ()	Get user name
bool SetFullName (string fullName)	Set user name
string GetDescription ()	Get user information
bool SetDescription (string description)	Set user information
bool ChangePassword (string old, string new)	Change user password
bool CanChangeUsageRate (string rateType)	Get usage changing permission
bool CanChangeQuota (string quotaType)	Get quota changing permission

Database Table

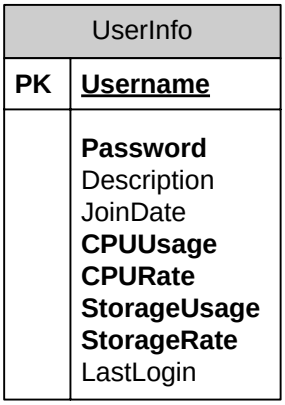

Figure 27 The database table of User Management Service

Virtual Computer Management Service (VCMS)

Components and interactions

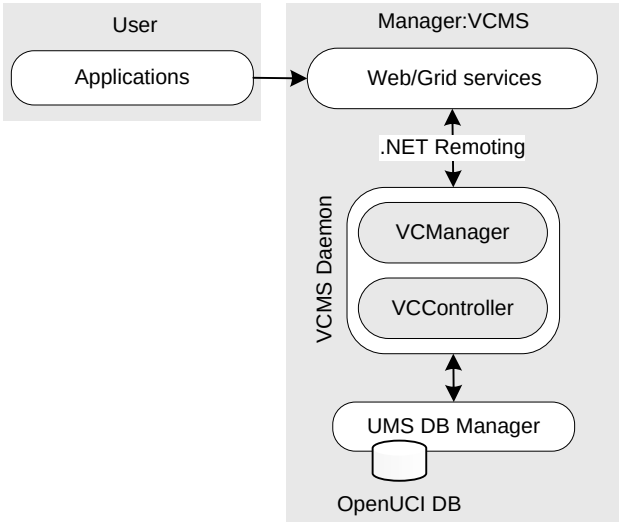

Figure 28 The component of Virtual Computer Management Service and its interactions

The purpose of the Virtual Computer Management Service is to provide a set of API for creating and managing the virtual computing environment which must be automatically adjustable depended on its' workload.

1. Virtual Computer Manager (VCManager) provides API for building virtual computers. For example, the virtual computer can comprise of ten 3GHz processors, 100 GB storage, and already deployed 3D graphic rendering service.

2. Virtual Computer Controller (VCController) provide mechanism to automatically adjust the configuration of virtual computers by detecting the workload of all active virtual computers, and decide that which one should be increased or decreased computing power or storage.

Figure 29 shows the state of the Virtual Computer Management Service. After it was started, it will create two threads, the virtual computer manager (VCManager), and the virtual computer controller (VCController) threads. The VCManager thread will wait for a management commands such as create, add, delete, start, and stop a virtual computer. When it received a management command, it will check and process command according to the type of command. For the VCController thread. After it was started, it will periodically check the job queue of each active virtual computer. If the number of job in queue is more than the maximum queue length threshold, it will discover for a suitable worker, and add it to the virtual computer. In the same way, if the number of job in queue is less than threshold, the VCController will remove some worker from the virtual computer.

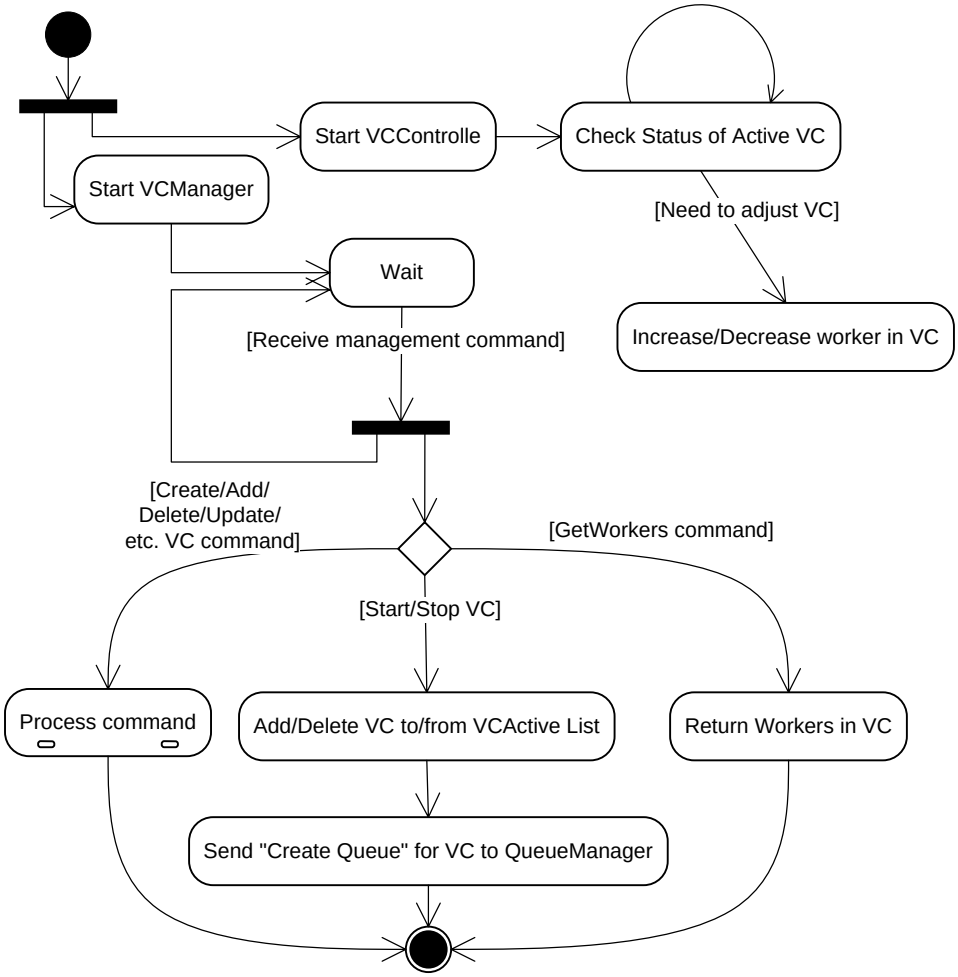

Figure 29 The state diagram of Virtual Computer Management Service

The API list

Table 7 The Virtual Computer Management Service API list

Functions	Description
`VritualComputer CreateVC (string name, int workerSpeed, int minWorkers, int maxWorkers, string[] services, bool allowNondedicated, bool autoAdjust)`	Create virtual computer
`bool SaveVC (VirtualComputer VC )`	Save virtual computer
`bool DelVC (string VCName)`	Delete virtual computer
`bool UpdateVC (VirtualComputer VC)`	Update virtual computer
`VirtualComputer GetVC (string VCName)`	Get virtual computer
`VirtualComputer[] ListVC ()`	List all virtual computers
`string GetDefaultVC ()`	Get default virtual computer
`bool SetDefaultVC (string VCName)`	Set default virtual computer
`bool StartVC (string VCName)`	Start virtual computer
`bool ShutdownVC (string VCName)`	Shutdown virtual computer
`string[] GetVCWorkers (string VCName, int number)`	Get workers from virtual computer

Database Table

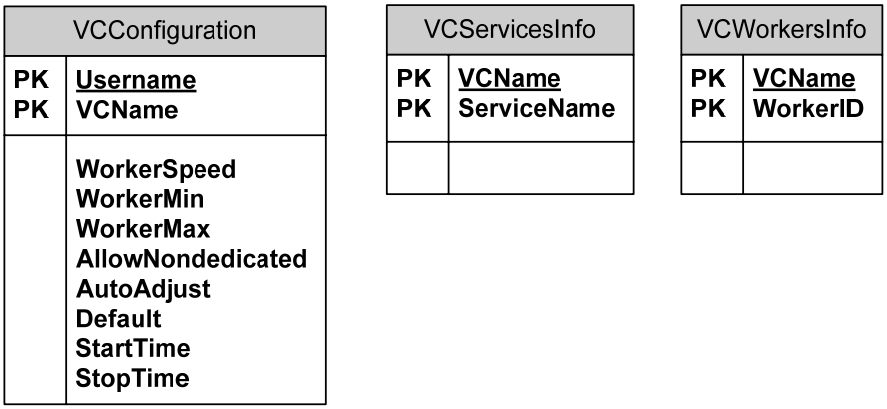

Figure 30 The database table of Virtual Computer Management Service

Job Management Service (JMS)

Components and state diagram

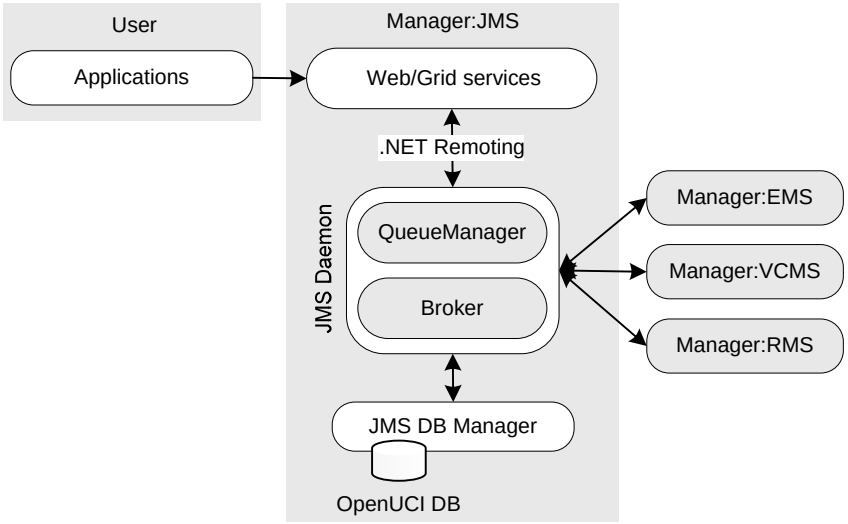

Figure 31 The component of Job Management Service and its interactions

The Job Management Service consists of two important components, QueueManager, and Broker, which are responsible for queuing, scheduling, selecting the workers, and dispatching jobs to the execution module (EMS). JMS also provides a set of API for creating, submitting, and operating jobs.

1. Queue Manager has two types of job queue, the global job queue, and the virtual computer job queue (VC queue). The global job queue is sued to store jobs that are not run on any virtual computers, and the VC queue is an individual queue for each virtual computer. The jobs in VC queue will be run on the workers in its' virtual computer.

Figure 32 shows the state of the Queue Manager. After the queue manager was started, it will wait for three types of commands, job submission from users, job request from broker, and queue creation from the virtual computer manager.

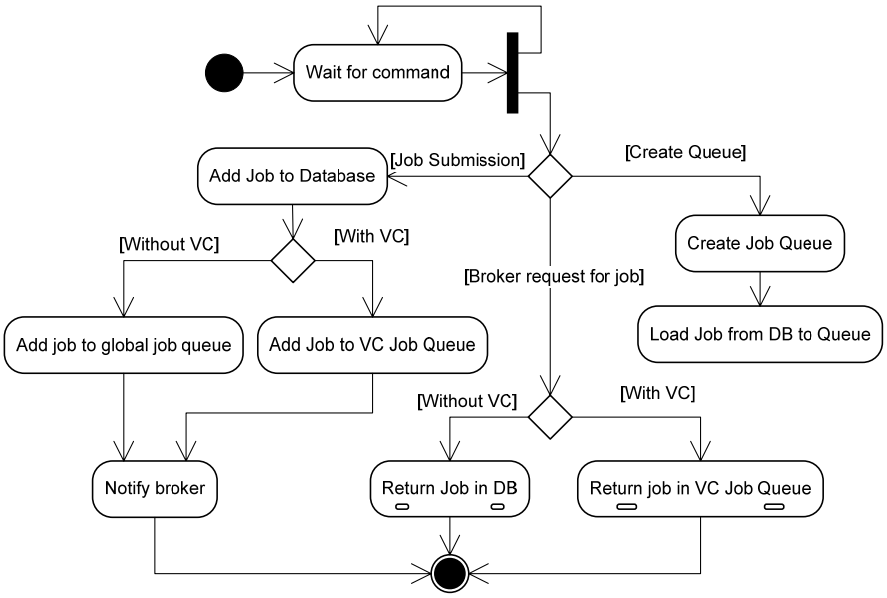

Figure 32 The state diagram of QueueManager

2. Broker is responsible for request a job from the queue manager, discover for the suitable worker to run the job, and send the job to the execution management service. The state of broker is shown in Figure 33 .

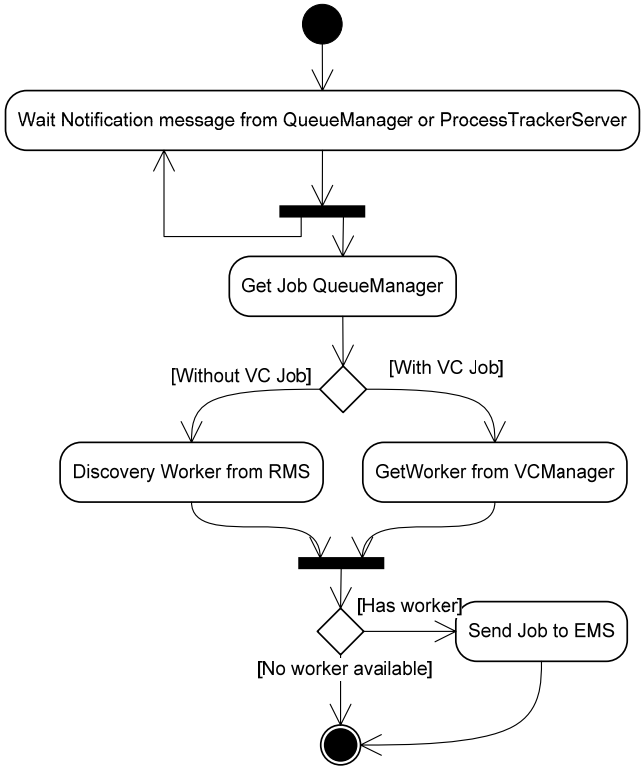

Figure 33 The state diagram of Broker

The API list

Table 8 The Job Management Service API list

Functions	Description
`UCIJob CreateJob (string jobType, string jobName, object result, string stdin, string stdout, string exeFile, string[] args)`	`Create OpenUCI job`
`UCIBatchJob CreateBatchJob ()`	`Create batch jobs`
`bool AddJobToBatchJob (UCIBatchJob bJob, UCIJob job, UCIJob dependency)`	`Add job to batch`
`bool DelJobFromBatchJob (UCIBatchJob, string jobName)`	`Delete job from batch`
`bool SubmitJob (UCIJob job, string VCName)`	`Submit job`
`bool SubmitBatchJob (UCIBatchJob bJob, string VCName)`	`Submit batch jobs`
`bool CancelJob (int jobID)`	`Cancel job`
`bool PauseJob (int jobID)`	`stop job`
`bool ResumeJob (int jobID)`	`Resume job`
`object GetJobResult (int jobID)`	`Get result of job`

Database Table

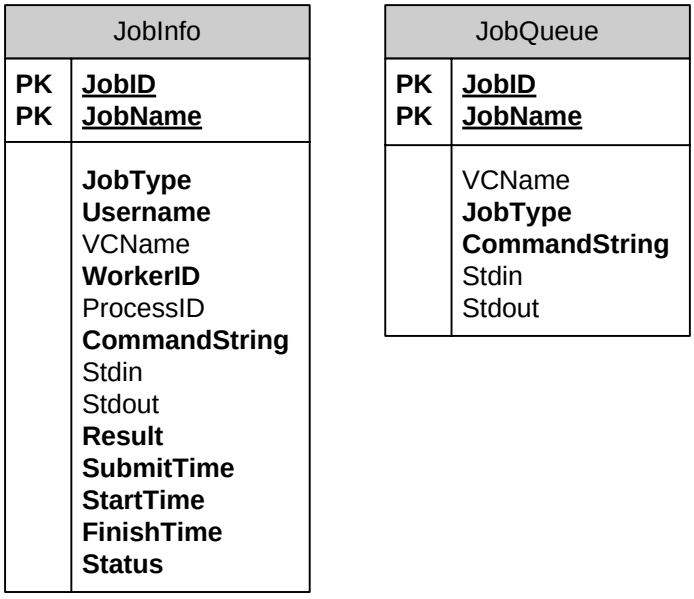

Figure 34 The database table of Job Management Service

The interaction of all core services

Figure 35 shows the interaction of user application, manager core services and worker agents. The user application has to authenticate to OpenUCI system using UMS service before using other services.

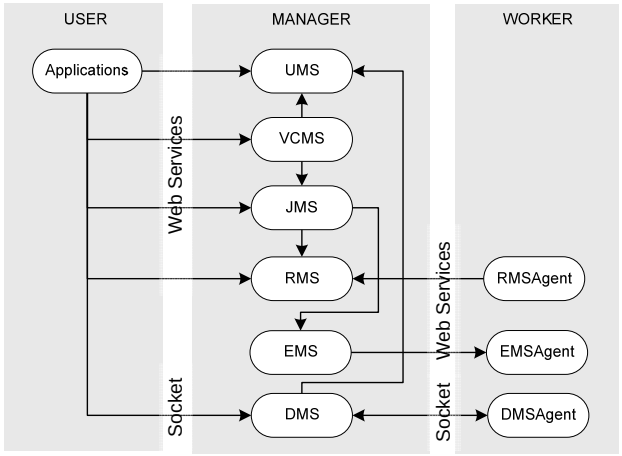

Figure 35 The interaction of all core services

Prototyped Framework Implementation

In this section, we describe about the technologies and pattern for the already developed OpenUCI core services in the prototyped version.

OpenUCI is developed using Microsoft C# language. Every core service in OpenUCI uses .NET web services as an interface for communication with any other systems or applications, and we use .NET remoting and socket techniques for internal communication. Figure 36 shows the design pattern of OpenUCI core services which follows the three tiers software architecture design.

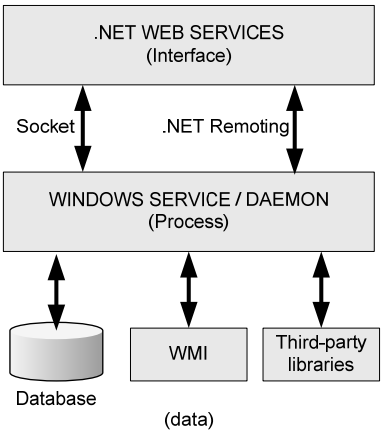

Figure 36 The design pattern of OpenUCI core services

Figure 37 shows a set of currently implemented core services. These include:

1. Resources Management Service (RMS)

    - Resources Collecting (RC)
    - Resources Discovery (RD)

2. Execution Management Service (EMS) (this service is implemented in another related project)

    - Executable file launcher (EXLauncher)

3. Job Management Service (JMS)

    - Broker

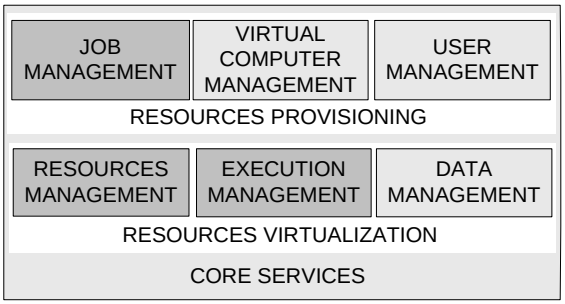

Figure 37 The implemented core services

In the future, more core services can be easily added to the system.

RESULTS AND DISCUSSION

Testing Software

Currently, the high performance computing is widely needed not limit to the computer research field anymore. The financial engineering (FE) is one field that need 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 evaluate the performance of OpenUCI system by applying the existing financial engineering application named Value-at-Risk (VaR) calculation which was implemented in .NET web services. The VaR measures the maximum money v that we will lose in n days of holding a set of bonds.

In this test, we use 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 for VaR web services, and then invoke them.

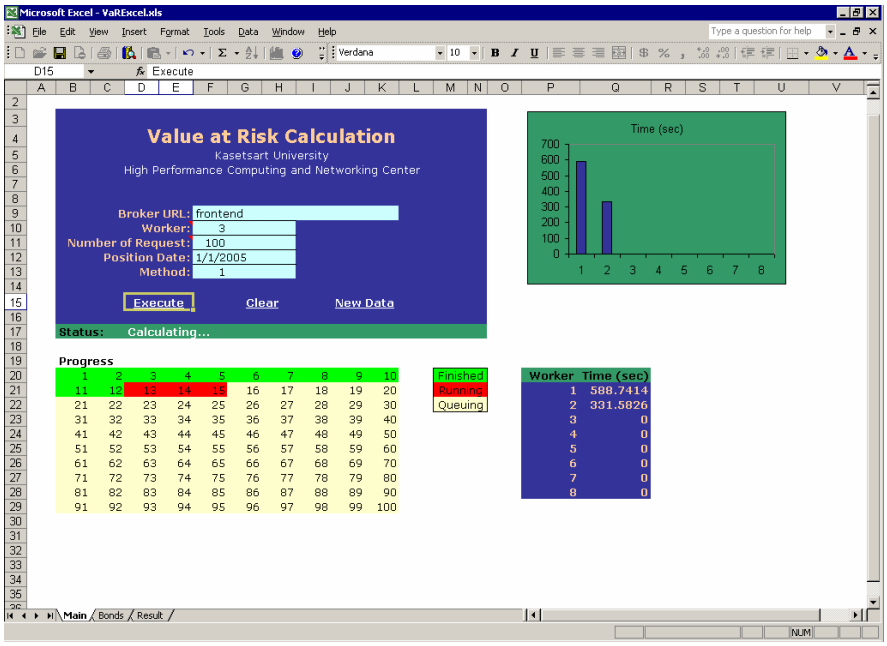

Figure 38 The VaR Excel client program

Test Configuration

The topology of test system is shown in Figure 10 in the materials and method chapter. The software that is installed on each machine is shown in Table 9 .

Table 9 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

Test Assumptions

1. Worker can execute job only one job at the time.

2. The input data is already in the workers

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 39 shows the procedure of this testing.

1. The client application discovery for a 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 to connect and invoke 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

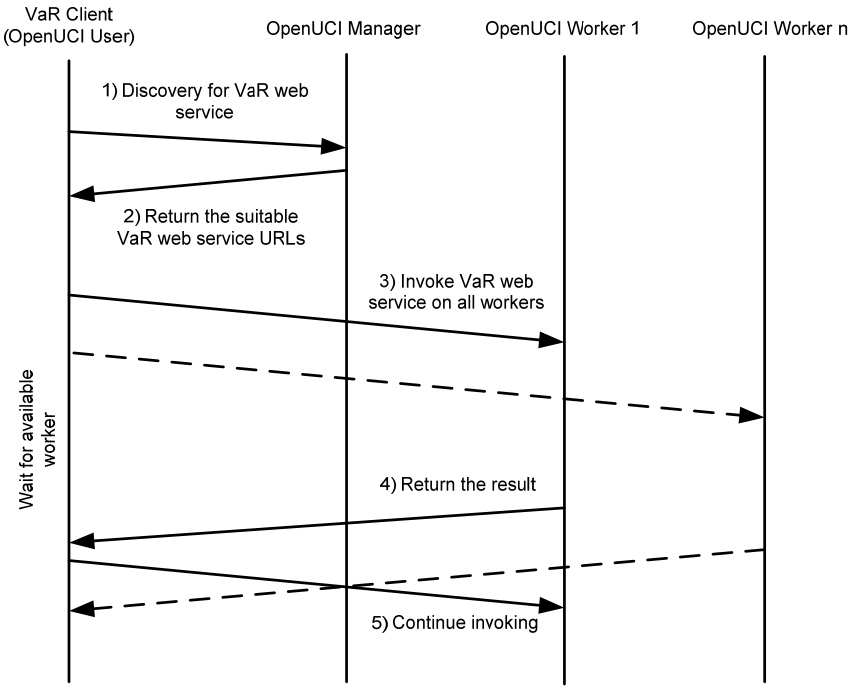

Figure 39 The throughput test procedure

The result of throughput test is shown in Table 10 , and Figure 40 . Figure 41 is the average of throughput of OpenUCI system which varies the number of workers.

Table 10 The throughput of OpenUCI

Time	1 Worker	2 Workers	4 Workers	8 Workers	16 Workers	32 Workers
10	0.2	0.3	0.7	1.4	3.1	6
30	0.2333333	0.4333333	0.7333333	1.633333	3.166667	6.266667
60	0.2166667	0.4166667	0.8166667	1.566667	3.183333	6.333333
90	0.2111111	0.4111111	0.7888889	1.644444	3.244444	6.444445
120	0.2166667	0.425	0.8416666	1.666667	3.308333	6.466667
180	0.2111111	0.4166667	0.8277778	1.627778	3.3	6.558333
240	0.2125	0.4208333	0.8375	1.670833	3.308333	6.5625
300	0.21	0.42	0.8366666	1.646667	3.306667	6.603333

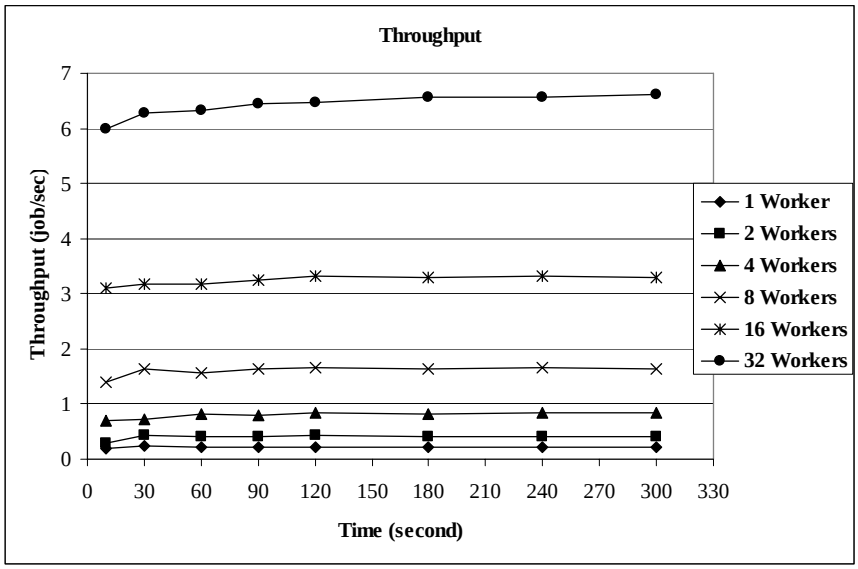

Figure 40 The throughput of OpenUCI system

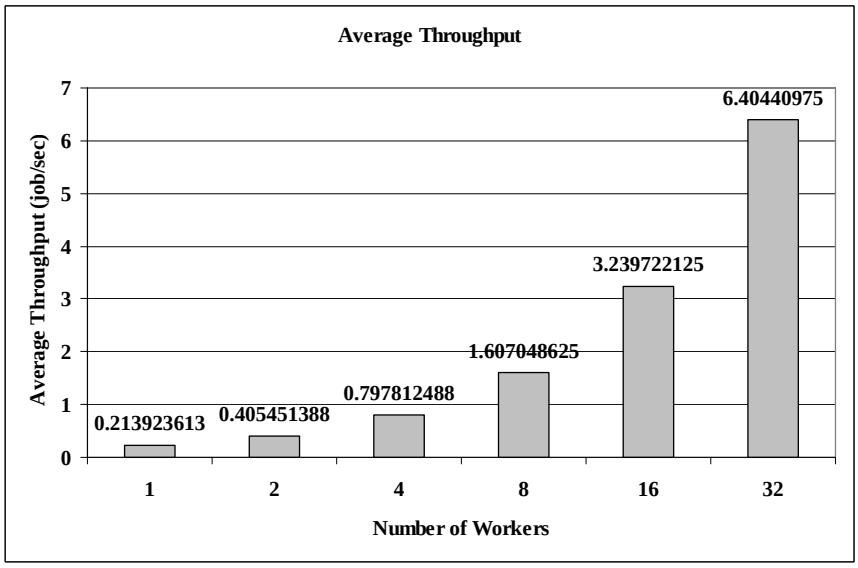

Figure 41 The average throughput of OpenUCI system

From the result, it showed that OpenUCI system still give a good throughput when then number of workers increased, and the increasing of throughput is nearby the increasing of number of workers. For example, the average throughput of 32 workers system is ~6.4 job/sec and the average throughput of 1 worker system is ~0.214 job/sec. the throughput was increased about 30 times

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 is similar to the throughput testing, but the speed up test will change the number of jobs submitted to system and observe the run time instead of fixing the run time and observe the number of finished jobs.

Table 11 and Figure 42 show the run time of this testing. Table 12 and Figure 43 show the speed up. Table 13 and Figure 44 show the efficiency.

Table 11 The run time of testing (second)

Worker	100 Jobs	500 Jobs	1000 Jobs	2000 Jobs	3000 Jobs
1	476.328	2359.171	4726.765	10083.328	14794.656
2	248.031	1191.593	2400.578	4734.843	7106.318
4	122.14	596.765	1185.054	2386.187	3566.531
8	61.818	303.696	609.306	1216.188	1825.014
16	33.297	157.656	308.283	619.43	923.287
32	19.875	76.25	151.921	301.968	451.546

Table 12 The speed up of testing

Worker	100 Jobs	500 Jobs	1000 Jobs	2000 Jobs	3000 Jobs
1	1	1	1	1	1
2	1.92043736	1.97984631	1.96901121	2.12960134	2.08190177
4	3.89985263	3.95326636	3.98864946	4.22570737	4.14819218
8	7.70532855	7.76819912	7.75762097	8.29092871	8.10659863
16	14.3054329	14.964042	15.3325516	16.2783979	16.0238972
32	23.9661887	30.9399475	31.1133089	33.3920415	32.7644492

Table 13 The efficiency of testing

Worker	100 Jobs	500 Jobs	1000 Jobs	2000 Jobs	3000 Jobs
1	1	1	1	1	1
2	0.96021868	0.98992315	0.98450561	1.06480067	1.04095088
4	0.97496316	0.98831659	0.99716237	1.05642684	1.03704804
8	0.96316607	0.97102489	0.96970262	1.03636609	1.01332483
16	0.89408956	0.93525262	0.95828447	1.01739987	1.00149358
32	0.7489434	0.96687336	0.9722909	1.0435013	1.02388904

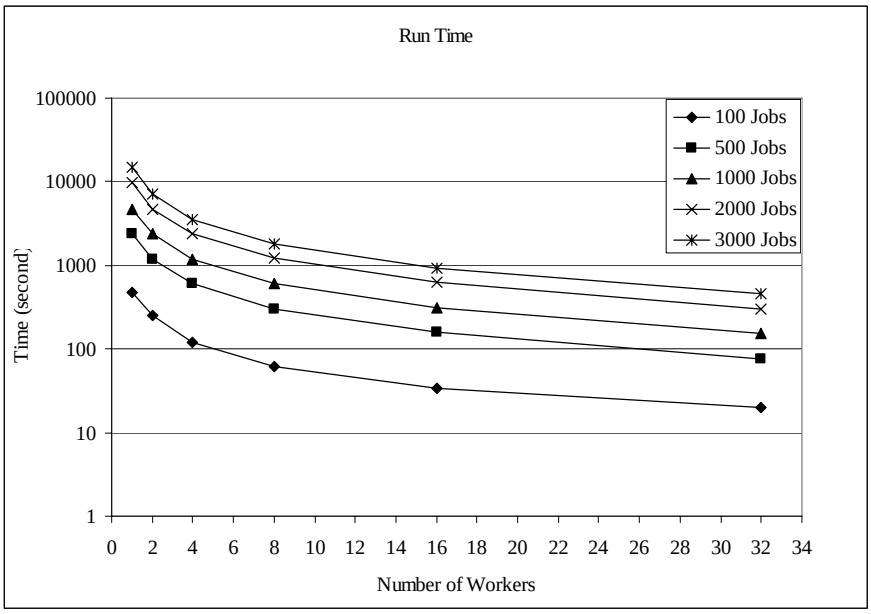

Figure 42 The run time plot

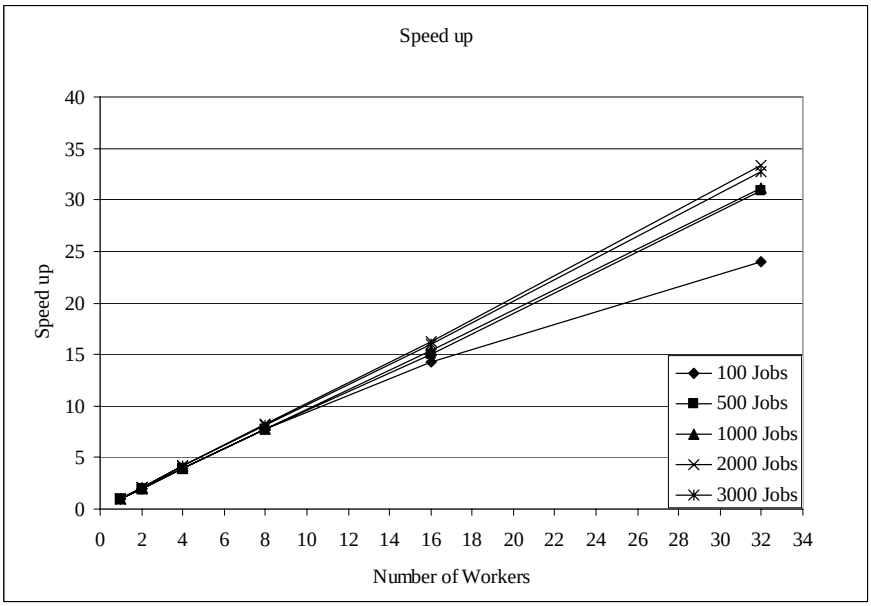

Figure 43 The speed up plot

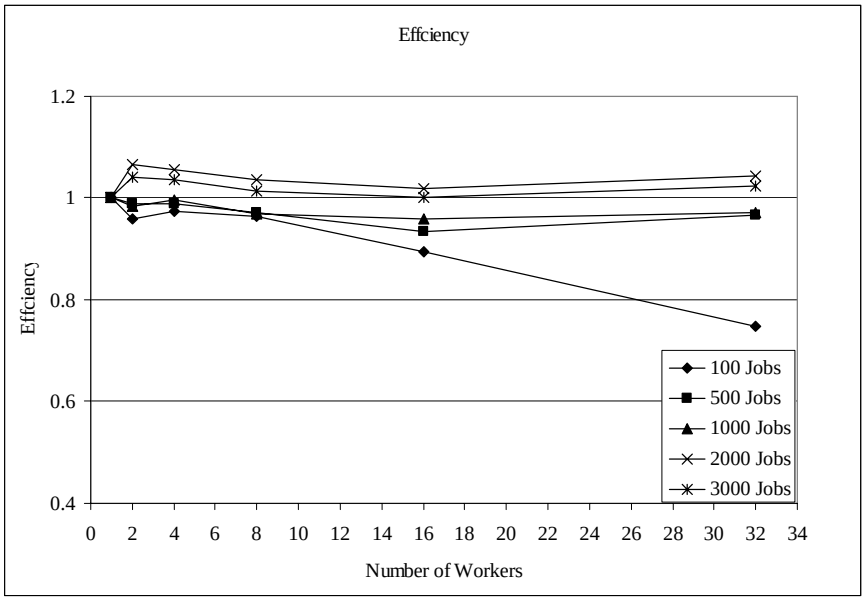

Figure 44 The efficiency plot

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 43 and Figure 44 , we found that there are three interested characteristic results.

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

2. The speed up and efficiency are almost perfect. The perfect speed up is the speed up that is equal to number of workers in system. The perfect efficiency is the efficiency that is equal to 1. Basically, the communication overhead such as input data transfer time makes the speed up and efficiency dropped, but, in this test, we reduce the data transfer time by replicating VaR database to all workers. So, the efficiency and speed up are 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 more number of jobs makes the total overhead time grower. However, the total overhead time will be reduced by the increasing of number of workers. So, at the a large amount of jobs such as 2000 and 3000 jobs, the run times of 2, 4, 8, 16, and 32 workers system are decreased more than the number of workers in system.

CONCLUSION

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 thesis proposed the design and implementation of framework used to build 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

Future Work

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.

1. Integrate the executable file launcher implemented in another related project to OpenUCI system

2. Implement the job queue management module

3. Implement the web and grid services invoker module

4. Implement the virtual computer management service

5. Implement the user authentication and accounting modules

6. Implement the data transfer service

7. Explore the mechanisms for handling fault of machines and jobs

8. Investigate a proper workload distribution scheme and study using simulation

LITERATURE CITED

Albaugh V. and H. Madduri. 2004. The utility metering service of the Universal Management Infrastructure. **IBM Systems Journal Vol. 43, NO. 1, 2004**

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

Chaisiri S. and P. Uthayopas. 2005. GUSTOFarm: A Software Framework for Constructing a Utility Computing Infrastructure. **International workshop on Applied Information Technology 2005 (IAIT 2005), November 25-26, 2005,** Thailand.

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,** Vol. 43, NO.1, 2004

Foster I. and C. Kesselman. 1998. **The Grid: Blueprint for a Future Computing Infrastructure**, Morgan Kaufman

_____, _____ and S. Tuecke. 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. **International Journal of Supercomputer Applications.** 15(3)

_____, _____, 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,** Vol. 42, NO.1, 2003

HP. 2006. HP Utility Data Center. **HP Utility Data Center – Solution Components.** Available source: http://h71028.www7.hp.com/enterprise/cache/259009-0-0-225-121.html, March 14, 2006.

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

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

Microsoft. 2006. The Drive to Self-Managing Dynamic Systems. **Dynamic Systems Initiative**. Available source: http://www.microsoft.com/windowsserversystem/dsi/default.mspx, March 14, 2006.

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.

_____, 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, 1992.**

Snelling D., I. Robinson and T. Banks. 2006. OASIS Web Services Resource Framework (WSRF) TC. **OASIS Web Services Resource Framework (WSRF) TC**. Available source: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf, March 14, 2006.

Sun Microsystems. 2006. Utility Computing Overview. **Utility Computing Overview**. Available source: http://www.sun.com/service/utility/overview.jsp, March 14, 2006.

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

_____. 2005b. Towards Open Grid Service Architecture. **OGSA – The Open Grid Service Architecture**. Available source: http://www.globus.org/ogsa/, 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/mambo/images/stories/CoreGRIDSpreadingExcellence/Presentations/coregrid-general-ccgrid2005.pdf, March 14, 2006.

Unger J. and M. Haynos. 2006. A visual tour of Open Grid Services Architecture. **IBM developer work**. Available source: http://www-128.ibm.com/developerworks/webservices/library/gr-visual/, March 14, 2006.

Vambeneqe W., P. Nibelet and S. Malaika. 2006. OASIS Web Services Notification (WSN) TC. **OASIS Web Services Notification (WSN) TC**. Available source http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn, March 14, 2006.

CURRICULUM VITAE

NAME:	Mr. Thanapol Rojanapanpat		
BIRTH DATE:	August 18, 1981		
BIRTH PLACE:	Bangkok, Thailand		
EDUCATION:	YEAR	INSTITUTION	DEGREE/DIPLOMA
	2002	Kasetsart University	B. Eng. (Computer Engineering)
POSITION/TITLE:	Graduate student		
WORK PLACE:	High Performance Computing and Networking Center, Faculty of Engineering, Kasetsart University		