# 國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

利用網格技巧改善一個大規模工作流程管理系統的回應時間

Improving the Response Time in a Large-scale WfMS with Grid Techniques

研 究 生：陳君豪

指導教授：王豐堅　教授

中 華 民 國 九 十 六 年 十 月

# 利用網格技巧改善一個大規模工作流程管理系統的回應時間

# Improving the Response Time in a Large-scale WfMS with Grid Techniques

研 究 生：陳君豪　　　Student：Chun-Hao Chen

指導教授：王豐堅　　　Advisor：Feng-Jian Wang

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

October 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年十月

# 利用網格技巧改善一個大規模工作流程管理系統的回應時間

研究生：陳君豪　　　　　指導教授：王豐堅 博士

國 立 交 通 大 學
資訊科學與工程研究所
碩 士 論 文

## 摘要

在現今的商業流程環境中，工作流程管理系統多是集中式的主從架構。當使用者對伺服器的請求大量增加時，造成請求的回應時間增加。在此狀況下，集中式的單一伺服器會形成效能瓶頸。為了解決此問題，本篇論文提出了一個利用網格技巧使其具備擴展性的工作流程平台。它可依需求動態增減資源的使用，維持可接受及穩定的請求回應時間。最後，我們利用這個架構來實作了一個原始系統(prototype)以及對此系統進行了一連串的效能測試，結果驗証了處在不同的使用者請求量下，整體系統皆能將請求的回應時間維持在能可接受及穩定的狀態。

**關鍵字：** 工作流程管理系統、網格、依需求動態調整、可擴展性

# Improving the Response Time in a Large-scale WfMS with Grid Techniques

Student: Chun-Hao Chen Advisor: Dr. Feng-Jian Wang

Institute of Computer Science and Engineering

National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

## Abstract

In the current business process environment, almost all workflow management systems are based on the centeralized client/server architecture. The request response time will increase greatly when the requests arrive at the PASE server at a high rate. In such circumstances, the single centralized server becomes the performance bottleneck. In order to solve this problem, thesis proposes a grid-enabled workflow computing platform. It can dynamically add or remove resources on demand, and maintain acceptable and stable request response time. We implemented the prototype system based on the proposed architecture and conducted a series of experiments for performance evaluation. The results of experiments evidence that under different workloads, ranging from 50 to 2,500 instances, the proposed architecture can deliver a nearly constant response time, benefiting from its scalable feature.

**Keywords:** Workflow Management system, grid, on-demand, scalability

# Table of Contents

摘要 .......................................................................................................................I

Abstract.....................................................................................................................II

Table of Contents ..................................................................................................... III

List of Tables .......................................................................................................... III

List of Figures…………………………………………… ……..................................III

1. Introduction ............................................................................................................. 1

2 Background…………………..……………………………………………………3

2.1 Agentflow: A Wrokflow Management System………………………………………3

2.2 Grid Computing for Dynamic Resource Provioioning………………………………5

2.2.1 Types of Grid Computing…………………………………………………...5

2.2.2 The evolution of the Grid Computing………………………………………6

2.2.3 The Grid Architecture………………………………………………………8

2.3 Grid Workflow System……………………………………………………………9

3. A Grid-Enabled Scalable Workflow Computing Platform ................................................ 11

3.1. The PASE Grid Architecture…………………………………………………..11

3.1.1. PASE Resource……………………………………………………………11

3.1.2 Process Definition Repository and Global Runtime Repository……………14

3.1.3 PASE Information Server…………………………………………………14

3.1.4 PASE Broker………………………………………………………………16

3.2. On-Demand Resource Provisioning Strategies……………………………………..19

3.2.1 User Request Processing…………………………………………………..19

3.2.2 Adaptable Resource Allocation……………………………………………21

4. Performance Evaluation ......................................................................................... 29

4.1. Experimental Settings……………………………………………………………29

4.1.1. PASE Resources……………………………………………………………..29

4.1.2 Process definitions and PDR………………………………………………....30

4.1.3 PASE Information Server…………………………………………………….31

4.1.4 PASE Broker………………………………………………………………….32

4.2. The Experiment Driver………………………………………………………...35

4.3. Experimental Result…………………………………………………………...37

5. Conclusions and future work........................................................................................ 44

References…………………………………………………………………………….46

# List of Tables

Table 3-1 PASE_Geninf……………………………………………………………………15

Table 3-2 PDR_Geninf……………………………………………………………………..16

Table 3-3 Instance id manipulation and request dispatching…………………………………19

Table 3-4 Returned id manipulation…………………………………………………………..19

Table 3-5 The Dynamic Request dispatching Algorithm for PER……………………………20

Table 3-6 PASE grid Performance Monitor Algorithm……………………………………….22

Table 3-7 Structure of PASEProperty………………………………………………………...24

Table 3-8 Adding Resource Algorithm……………………………………………………….24

Table 3-9 Removing Resource Algorithm……………………………………………………26

Table 3-10 Suspending Checking Algorithm…………………………………………………27

# List of Figures

Fig 2-1 Agentflow System Overview…...........................................................................4

Fig 2-2 The Grid Architecture……………………………………………………………9

Fig 3-1 The PASE grid architecture…………………………………………………….10

Fig 3-2 The architecture of PASE broker……………………………………………….17

Fig 4-1 PASE_Geninf…………………………………………………………………...31

Fig 4-2 PDR_Geninf……………………………………………………………………32

Fig 4-3 PIS Configurations……………………………………………………………...32

Fig 4-4 Performance Monitor Configurations…………………………………………..33

Fig 4-5 Select initial PASE resource……………………………………………………34

Fig 4-6 System status……………………………………………………………………34

Fig 4-7 Experiment Driver………………………………………………………………36

Fig 4-8 Results of createProcess()………………………………………………………37

Fig 4-9 Results of startTask()……………………………………………………………38

Fig 4-10 Results of completeTask()…………………………………………………….38

Fig 4-11 Results of getTaskOfCompany……………………………………………...39

Fig 4-12 Average response time of all requests………………………………………40

Fig 4-13 Performance results of createProcess()……………………………………….41

Fig 4-14 Performance results of startTask()……………………………………………42

Fig 4-15 Performance results of completeTask()………………………………………42

Fig 4-16 Perofrmance results of getTaskOfCompany…………………………………43

Fig 4-17 Average response time of all requests……………………………………...43

# Chapter 1. Introduction

To manage and automate business processes, workflow management systems (WfMS), have been broadly adopted by many enterprises to efficiently control the flow of tasks, assign the needed human resources and the needed artifacts for executing each task, and monitor the executions of tasks. Most current workflow management systems are client-server architecture, adopting a centralized workflow engine and a database server used to store process definitions and runtime data. For example, Agentflow system [4] Flowring Technology Corp, Agentflow system, http://www.flowring.comis a well-known java-based workflow management systems in Taiwan, is based on the centralized client-server architecture.

Obviously, the request response time in such a centralized client-server architecture is bounded by the computing power of the single centralized server and the capacity of the database server. The response time will increase greatly when the requests arrive at the PASE server at a high rate. In such circumstances, the single centralized server becomes the performance bottleneck.

Grid computing [1][2][3] has been under development and evolvement for many years. It enables users to access resources across different administrative domain, and aggregate those resources to solve some problems which otherwise can not be effectively solved on the resources inside an single administrative domain. To solve the performance bottleneck in Agentflow system, we extended it to a scalable workflow computing platform based on PASE grid architecture, which is proposed and described in detail in this thesis.

The rest of this thesis is organized as follows. Chapter 2 introduces Agentflow system, and grid computing. A scalable workflow computing platform and on-demand resource provisioning strategies are described in Chapter 3. Chapter 4 illustrates a series of experiments evaluating the PASE gird architecture and presents discussions of the experiment results. Chapter 5 concludes the thesis and points some future research directions..

# Chapter 2. Background

## 2.1 Agentflow: A Workflow Management System

Agentflow system [4] developed by the Flowring technology corporation is a java-based workflow management system based on the centralized client-server architecture. There are three main components in Agentflow, including PDE, Flow Engine and Agenda.

- PDE (Process Definition Environment) is a graphical editor for modeling different views of a business, including process view, artifact view and organization view. Each view is modeled by tools in PDE separately, including an Organization Designer for constructing the organization view, an e-form Designer for designing the artifact view, and a Process Designer for modeling process view.
- Flow Engine (also called PASE server) is a workflow enactment environment, which drives the flow of works and facilitates process enacting, control, management, and monitoring.
- Agenda is a client-side tool. The users can browse their own task-list, deal with what they have to do, initiate processes, monitor the states of the flow through the Agenda

An overview of the main components and their relationships in Agentflow is shown in Figure 2-1

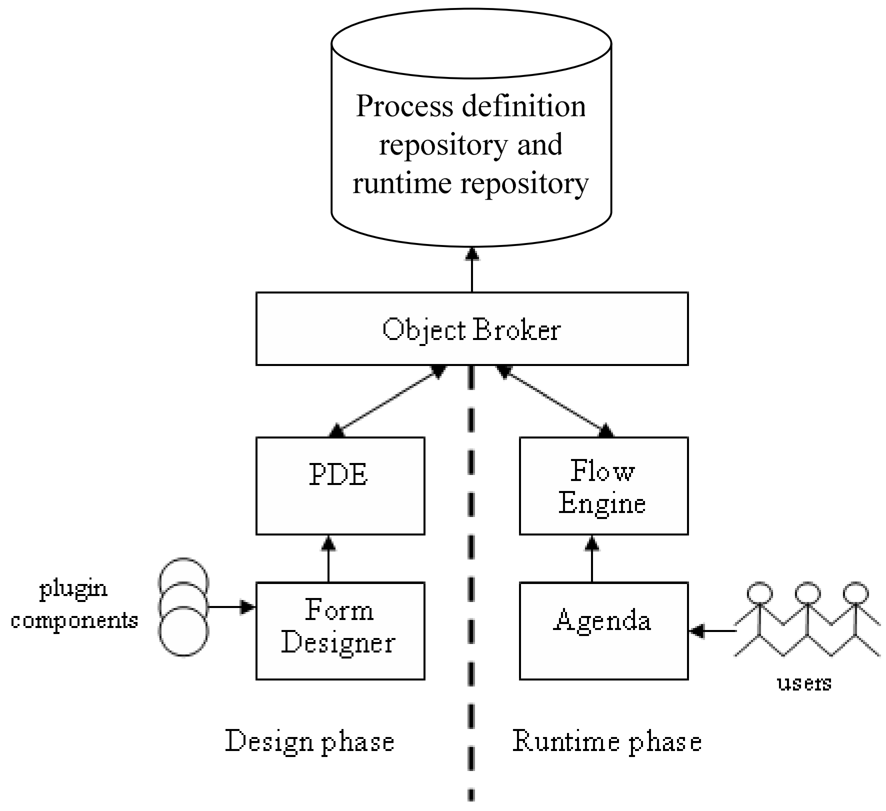


Figure 2-1. Agentflow System Overview

The database of Agentflow system contains two repositories, process definition repository and runtime repository. The process definition repository is used for storing process definitions, while the runtime repository is used for storing all the workflow instance data. In addition, Agentflow system provides a java-based interface, Workflow Common Interface (WFCI), which allows users to interact with the PASE server. For example, WebAgenda is a web-based agenda which communicates with PASE server through the WFCI.

## 2.2 Grid Computing for Dynamic Resource Provisioning

In recent years, the computer networks have been evolved rapidly and become more and more cheaper and faster. This trend contributes to the rapid development of grid computing technologies. Grid computing [1][2][3] is a distributed computing architecture, increasingly adopted by both scientific and business domains. Although, until now, there is no commonly agreeable and precise definition of what a grid is or what components are needed to construct a grid. Most people think of grid computing as a promising technology for providing a scalable, secure, and high-performance computing platform through automatically discovering and integrating geographically distributed resources [2].

### 2.2.1 Types of Grid Computing

In general, there are five major types of grid computing [1] which are described as follows:

- Distributed supercomputing

  The characteristics of distributed supercomputing applications are that they need to solve very large problems, like stellar dynamics, required a lots of CPUs, memories, etc. In order to fulfill these requirements, they use grid technologies to aggregate substantial computational resources.

- High-throughput computing

  Applications of this type need to complete large numbers of loosely coupled or independent tasks with high throughput, like chip design. Hence, they use grid tehcnologies to discovery, negotiate, and utilize the idle resources, and then schedule those tasks run on them.

- On-Demand computing

  The characteristics of on-demand computing applications are that they need resources for short-term requirements and those resources are costly and inconveniently located locally. Hence, they use grids for meeting their demands.

- Data-intensive computing

  This kind of applications focus on collecting large amount of new information from geographically distributed storage system, like sky survey, and then manipulating them. There are several data grid technologies which could achieve these requirement, such as Globus Data Grid [18] and Storage Resource Broker [15].

- Collaborative computing

  The characteristics of collaborative computing applications are that they support communication and collaborative work between multiple groups, like collaborative design. They use grids to provide a virtual shared data space Many collaborative computing applications focus on sharing computational resources rather than data resources, they may also have the characteristic of the others types of grid computing.

The scalable PASE grid architecture proposed in this thesis aims to deal with workflow computing requests which belong to the high throughput computing as well as the on-demand computing categories.

### 2.2.2 The Evolution of the Grid Computing

The three stages of the evolution of grid computing identified in [3] are described below. These generations are not strictly defined; they are distinguished by philosophies rather than technologies.

- The first generation

In the early to mid 1990s, the emerging technology, metacomputing [5], is the original concept of the grid computing. The goal of metacomputing is to provide the high performance computational resources by linking a number of supercomputer sites together for solving scientific problems. There are two main projects of the metacomputing including FAFNER [6] and I-WAY. [7].

- The second generation

The grid environments of this generation are typified by many of today's grid applications. They want to solve some issues, arising from the first generation which include heterogeneity, scalability and adaptability. Middleware is a better choice to address those issues. In grid environment, the middleware provides a set of services and hide the heterogeneity for users by defining the interfaces.

In addition, they suggested some design features guide the development of grid applications including 1) administrative hierarchy (for scalability), 2)communication services, 3)information services, 4)naming services,  5) distributed file systems and caching, 6)security and authorization, 7)system status and fault tolerance, 8)resource management and scheduling, and 9)User and Administrative GUI

Some grid-related projects in this generation of the grid computing include Globus Toolkit [18], Legion [19], which provide the essential services needed to constructs grid applications.

- The third generation

This stage shifts focus from large-scale computing to distributed collaboration and virtual organization [17].. The characteristics of the grids of this generation are that the grids increasingly adopt service-oriented model and pay more attentions to the metadata.

The service-oriented approach defines the interface of each service component which describes the availability and functionality. It can assemble grid resources into grid applications flexibly by those predefined interfaces. The metadata supports dynamic reconfiguration of grid environment, such as self-organization and self-healing. This will introduce the new extension of the grid computing, i.e. autonomic computing [8] Kephart J.O , Chess D.M, "The vision of autonomic computing", Computer, Vol:36, Issue: 1, 2003. The autonomic computing will be the next generations of the grid computing. .

### 2.2.3 The Grid Architecture

Figure 2-2 illustrates the layered grid architecture proposed by [2]. Included in he lowest layer, the fabric are physical resources of the grid, such as computers, storage systems, networks and sensors.

Above the fabric layer are the connectivity and resource layers. The connectivity layer contains the communication and authentication protocols. Communication protocols are used by resources to exchange data as well as communicate with each other and authentication protocols provide secure mechanism for verifying the identity of both users and resources. The resource layer also contains protocols that utilize the connectivity and authentication protocols to provide secure initiation, monitoring and control of resource-sharing operation.

The collective services layer contains protocols, services, and APIs that implement interactions across collections of resources. The services include: 1) Directory and broking services for resource discovery and allocation, 2)Monitoring and diagnostic services, 3)Data replication services, and 4)Membership and policy service for keeping track of who in the grid is allowed to access resources.

The topmost layer is the applications layer. Applications are constructed by using components in other layers and then can run on the grid.

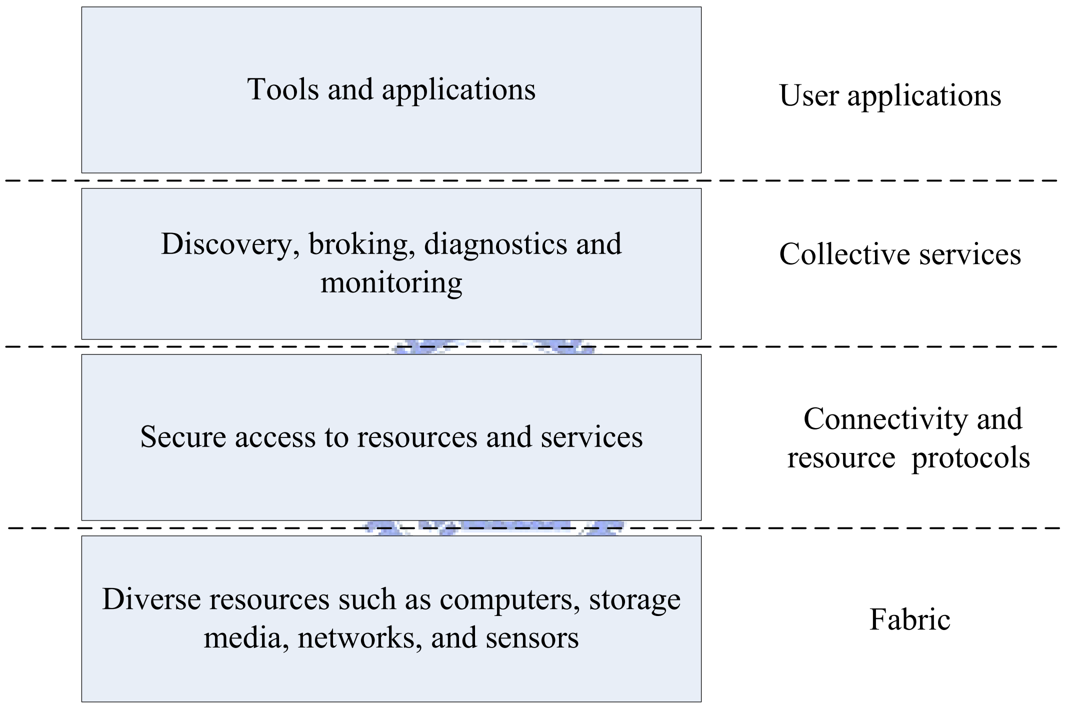

Tools and applications
User applications
Discovery, broking, diagnostics and monitoring
Collective services
Secure access to resources and services
Connectivity and resource protocols
Diverse resources such as computers, storage media, networks, and sensors
Fabric

Figure 2-2 The Grid Architecture

## 2.3 Grid Workflow System

The concepts of workflow are extremely important in grid computing. The systems manage the job dependencies and control the flows of jobs in a gird computing are called grid workflow system. Today, there are many grid workflow systems, such as GridAnt [9], Triana [10], XCAT [11], GridFlow [12], Kepler [13], and Grid-Flow..

All of the grid workflow systems mentioned above are facilitated to orchestrate grid-enabled programs or services. A common feature for these grid workflow systems is that they all provide a graphical user interface and a script language for users to model the workflow process.

# ■ Chapter 3. A Grid-Enabled Scalable Workflow Computing Platform

In this chapter, firstly, a grid-enabled scalable workflow computing platform based on Agentflow is introduced. This scalable platform produces acceptable and stable request response time under a wide range of request workloads. In addition to the system architecture, the strategies for achieving on-demand resource provisioning are also presented in the chapter.

## 3.1 The PASE Grid Architecture

The PASE server is a workflow enactment subsystem in an Agentflow system which drives the flow of works and facilitates process enacting, control, management, and monitoring. Because there is only one centralized PASE server with a dedicated database server in the original Agentflow system, the request response time will increase greatly when the requests arrive at the PASE server at a high rate. In such circumstances, the single centralized server for the platform becomes the performance bottleneck. In order to solve the performance issue, this thesis proposes a scalable workflow computing platform, PASE grid architecture, which extends a Agentflow system to a grid-enable system.

The following three concepts of grid computing guide us in designing the PASE grid architecture:

- Virtualization

  A PASE broker in the PASE grid provides a java-based interface, PASE broker

Common Interface (PBCI), which implements a set of functions supporting users to query and acquire PASE resources in the grid. Clients interact with PASE grid through the PBCI and need not know the underlying grid configuration such as the amount of PASE servers available and the computing speed of constituent machines. The PASE Grid thus like a powerful and scalable super-PASE server looks to the clients

- Utilizing resources across different administrative domains

  With the PASE Information server which will be described in details in section 3.1.3, resources located in different administrative domains could be dynamically integrated together to achieve a common goal, in some way realizing the concept of virtual organization

- On-demand resource provisioning

  When all the existing PASE servers have been overloaded, the PASE grid would automatically discover more computing resources and appropriately configure them to become newly available PASE servers to share the request workloads. On the other hand, when the incoming requests decrease and the overall system has been under-utilized, the PASE grid will remove a portion of the PASE resources to allow them to be utilized by other demanding PASE brokers or other applications.

The PASE grid architecture is shown in Figure 3-1, the components in which will be elaborated in the following sub sections.

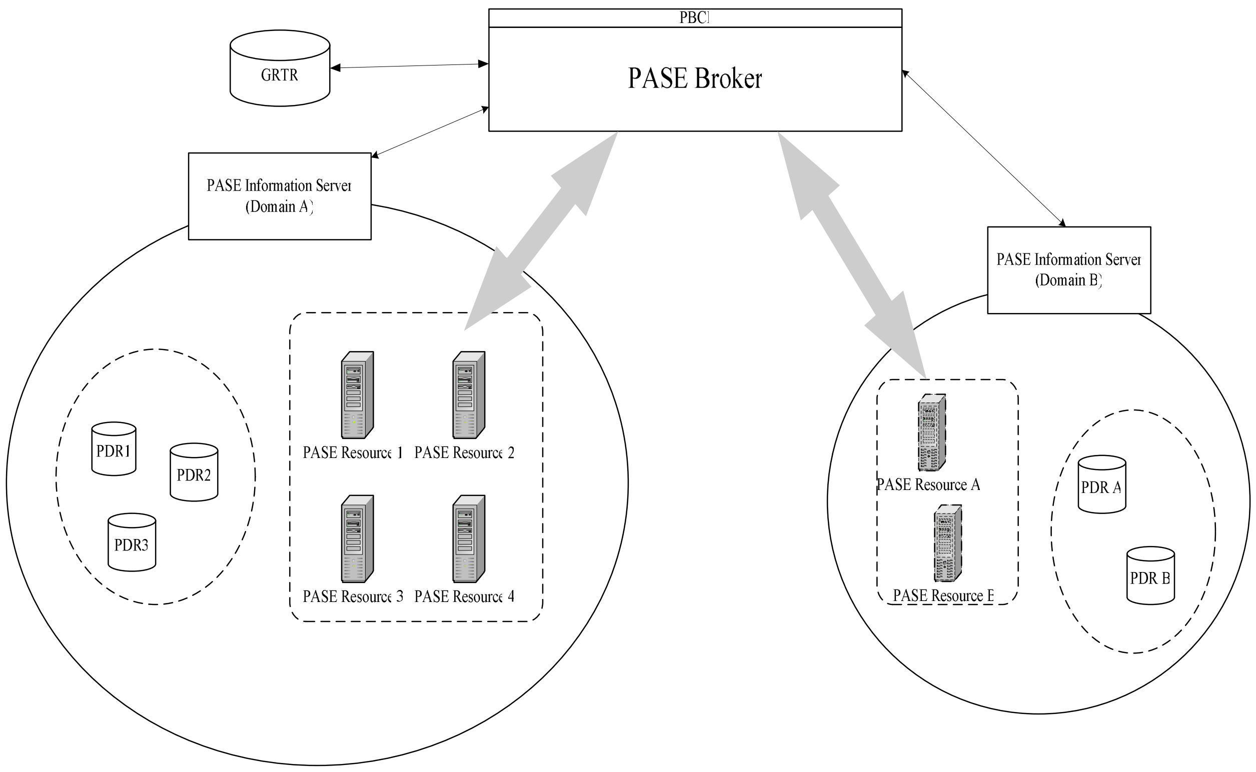


Figure 3-1 The PASE grid architecture

## 3.1.1 PASE Resource

A PASE resource is composed of software and hardware resources. The software resources include a PASE server which is a flow engine derived from the Agentflow system and a database used to store runtime data for PASE server and replicas of process definitions. The hardware resource is typically a computer like PC, notebook, or workstation on which the software resources can run. The PASE server and database of the PASE Resource can run on one or more site. Each PASE resource is managed by one *PASE information server* (PIS), and it can be used by only one PASE broker at any instant.

A PASE resource must be registered to the PIS in its domain before it can be included into the PASE grid. When a resource provider wants to withdraw the PASE resource provided by him, he must ask the PIS to delete the related record of the PASE resource from its database.

### 3.1.2 Process Definition Repository and Global Runtime Repository

The process definition repository (PDR) contains some business process definitions designed by process designer using process definition editor (PDE). When the PASE broker needs to add a new PASE resource, the PIS will replicate the content of PDR into the database of the new PASE resource according to the incoming request. In each domain, there can exist more than one PDR, and each PDR can be accessed by more than one PASE resource. The administrators are responsible for registering the PDR into the PIS.

The global runtime repository (GRTR) contains those workflow instances which have finished their execution. The completed workflow instances are kept in the repository for future references. When the PASE broker wants to remove a PASE resource, it will move the PASE resource’s runtime data into the GRTR. There is only one GRTR in the PASE grid, which is managed by the PASE broker.

### 3.1.3 PASE Information Server

The *PASE information server* (PIS) plays a role, similar to the MCAT in Storage Resource Broker [15] or Grid Information Service (GIS) in Globus Tookit [14], which is used to maintain necessary information about a domain and all the PASE resources in the domain. Furthermore, it is also responsible for replicating data from the PDR into new PASE resource and clearing the database of removed PASE resource..

The following tables describe the information maintained by PIS, such as PASE general information (PASE_Geninf), and process definition repository general information (PDR_Geninf). The information is required to assist the PASE broker in accessing, discovering, monitoring, and managing PASE resources.

Table 3-1 PASE_Geninf

| Attribute | Description |
|---|---|
| PASE_ID | The unique id of PASE Resource |
| PASE_Host | The host of the PASE Resource |
| PASE_Port | The port of the PASE Resource |
| Database_Name | The database name of the PASE Resource |
| Database_Host | The host of the database of the PASE Resource |
| Database_Port | The port of the database of the PASE Resource |
| Database_User | The user name and password grant for database accessing |
| Database_Password | |
| PDR_Id | The id of PDR to which the PASE can refer |
| State | The state of the PASE Resource. |
| Load_Max_Limit | The limit on the load (instances) |
| ArrivalRate_Max_Limit | The limit on the arrival rate |

. Table 3-1 illustrates the general information of PASE resource. The unique id is composed of (host:port). The state of PASE resource can be ready, reserving, running, or blocking. The ready state stands for the PASE resource being available, i.e. the database of the PASE Resource is already created, and the PASE server of PASE resource is started. The reserving state represents that the PASE resource is reserved by PASE broker, but it is not connected to the PASE broker yet. The running state stands for the PASE broker being using the PASE resource for serving incoming requests. The blocking state stands for the failure of the PASE resource.

‘

Table 3-2 PDR_Geninf

| Attribute | Description |
|---|---|
| PDR_ID | The unique id of the PDR |
| PDR_Name | The database name of the PDR |
| PDR_Host | The host of the PDR |
| PDR_Port | The port of the PDR |
| PDR_User | The user name and password grant for PDR accessing |
| PDR_Password | |

Table 3-2 describes the general information of PDR. The unique id of PDR is start as “PDR”. When an administrator registers a new process definition repository into the PASE grid, the PIS will generate this information according to the properties of the PDR.

### 3.1.4 PASE Broker

The PASE broker is a vital part of the PASE grid; it is responsible for coordinating the PISs, PASE resources process definition repositories, and the global runtime repository. It can manage multiple PISs and use the PASE resources belonging to those PISs. The architecture of the PASE broker is represented in Figure 3-2.

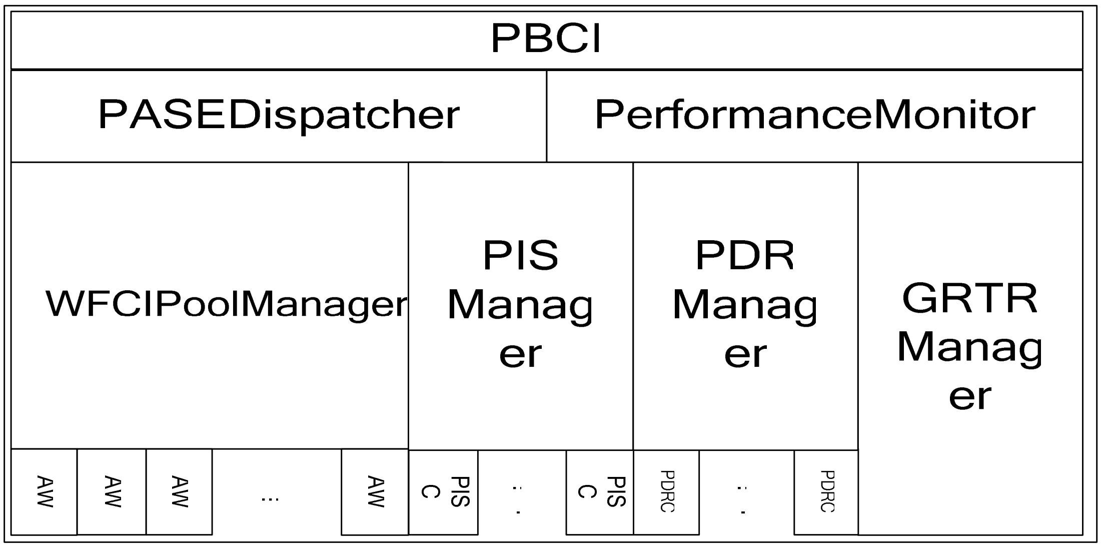


Figure 3-2 The architecture of PASE broker

*PISManager* is responsible for managing the PIS connections (PISCs) to all PISs. It retrieves and caches the information maintained in PISs. Initially, the administrator can select the PASE resources and the PDRs he/she wants to used, then the PISManager send replication request to PISs for replicating process definitions into each PASE resource.

The *PDRManager* manages the PDR connection (PDRCs) to all PDRs. The requests from clients that want to get the process definition relevant data are manipulated by the PDRManager. The *GRTR Manager* backups the workflow instance from the PASE resource which is to be removed by the PASE broker.

The *WFCIPoolManager* creates *AbstractWFCIs (AWs)* to connect to those selected PASE resources with the RMI mechanism. The AW is a component to wrap the WFCI connection and records some metadata about the WFCI connection, such as a list of processes and a list of member records. In addition, AW measures some metrics, like load, average arrival rate of requests, and average response time of requests.  Those metrics are as the basis used by PerformanceMonitor for monitoring performance.

The WFCIPoolManager contains three kinds of pools corresponding to different states of AWs, including running pool, suspending pool, and blocking pool. The running pool contains the healthy AWs. The suspending pool contains AWs which would not take any new create-process requests but still have some unfinished workflow instances running on it. The blocking pool contains AWs which run into some kinds of failure founded by the PASE broker.

The *PerformanceMonitor (PM)* monitors the performance of the overall system by different load determination modes. These modes include load (instance), the average request arrival rate and the average request response time. When the system is overloaded, it informs the *WFCIPoolManager* to find out new available PASE resources from PISs and create connections to them. If there are no new PASE resources found, it sends an alert to administrators and they can add new PASE resources manually. Moreover, when the system has been under-utilized for a specific period of time, it also informs the *WFCIPoolManager* to remove some AWs.

The *PASEDispatcher* does some pre-actions for each request to manipulate the some workflow instance relvant parameters, such as the identity of task (TskID), the identity of artifact instance (AnsID), the artifact instance (PASEartInstance), and the identity of attached file (FileID). It then selects an appropriate PASE resource and delegate the request to it Table 3-3 shows an example.

Table 3-3 Instance id manipulation and request dispatching

| InstanceID | Allocated Resource | ID in the Resource |
|---|---|---|
| Tsk(140.113.210.11:20000)000000000001 | 140:113.210.11:20000 | Tsk000000000001 |
| Ans(140.113.210.21:20000)000012345678 | 140.113.210.21:20000 | Ans000012345678 |

Before each request returns to user, the PASE broker does post-actions to append the identity of PASE resource to instance relevant data, and merge the return data from different PASE resources. Table 3-4 shows an example

Table 3-4 Returned id manipulation

| Returned ID | Source Resource | Appended ID |
|---|---|---|
| Tsk000000000001 | 140:113.210.11:20000 | Tsk(140.113.210.11:20000)000000000001 |
| Ans000012345678 | 140.113.210.21:20000 | Ans(140.113.210.21:20000)000012345678 |

## 3.2 On-Demand Resource Provisioning Strategies

This section discusses the resource provisioning strategies used in the PASE grid. There are two kinds of demand, each of which has different strategies. The following sections represent the detail of those strategies.

### 3.2.1 User Request Processing

The response time of each request is determined by the computing capabilities of the PASE server and the capacity of the database server in the centralized Agentflow system. The PASE grid architecture alleviates the performance bottleneck of the centralized server with its scalable computing capabilities, and thus produces shorter response time for user requests. Among different kinds of requests, *manipulate task* requests (MTRs) and *process enactment requests* (PERs) can benefit from this PASE grid architecture. On the other hand, the *collect data* requests (CDRs) would take a little bit longer time than in the original centralized architecture. Therefore, overall speaking, the proposed PASE grid architecture can effectively improve the runtime performance for most workflow activities. How each kind of requests is processed is illustrated in the followings.

- The PER is used to create a workflow instance according to a predefined process definition, such as createProcess() in the PBCI. When a PER occurs, a PASE resource is then selected for processing the request according to a dynamic request dispatching algorithm which is described in Table 3-5. The unique PASE resource will be provided for PER according to the metric selected by the administrator. The PER resource provisioning algorithm is represented in Algorithm 3.1.

Table 3-5 The Dynamic Request Dispatching Algorithm for PER

**Algorithm 3.1 (Dynamic Request dispatching for PER)**

**Input:**

```
    The user id U
    The process id  P
    The load determination mode M
```

**Output:**

```
    A candidate PASE resource id R
```

**PER_RP (U,P,M):**

```
01    Begin
02        // Get the list of AbstractWFCI which are running
```

```
    List wfciList=wfciPoolManager.getRunnongs();
    AbstractWFCI t=new AbstractWFCI();
    For each AbstractWFCI a∈ wfciList do
      // Compare the current workload of each PASE resource
      If (a.getMaxLoadByMode(M)-a.getLoadByMode(M))>
          (t.getMaxLoadByMode(M)-t.getLoadByMode(M))
          t=a;
      EndIf
    EndFor
    R = t.getID();
End
```

- The PASE resources required for the CDR and MTR are not determined by the dynamic request dispatching algorithm. The CDR is used to retrieve the instance relevant data or process definition related data, *e.g.* getTaskList() or getMemberRecrod() in the PBCI. A CDR may require more than one PASE resources to collaboratively accomplish its request and these PASE resources are determined by the data to be retrieved. The MTR is used to manipulate an task or a group of tasks, *e.g.* startTask(), suspendTask() and completeTask() in the PBCI. A MTR will be dispatched to the PASE resource where the workflow instance generating this request was created

### 3.2.2 Adaptable Resource Allocation

The PerformanceMonitor monitors the performance of each PASE resource in the PASE grid, it sends the event to the WFCIPoolManager when the entire PASE grid is overloaded or under-utilized. Table 3-6 describes the PASE grid performance monitoring algorithm.

Table 3-6 PASE grid Performance Monitoring Algorithm

**Algorithm 3.2 (Performance Monitoring Algorithm)**

**Input:**

```
    The monitoring interval I
    The list of running AWs L
    The load determination mode M
    /* When a continuous underutilized time period exceeds this predefined
       threshold, the system will remove some PASE resources. */
    A time period C
```

**PM (I,M,C,L):**

```
Begin
  long u_Time = 0;
  List pdrList;
  boolean isUnderUtilized=false;
  While(true) do
     sleep(I);
     int o_count=0;
     int u_count=0;
     For each AbstractWFCI a∈ L do
        If a.getLoadByMode(M)>a.getMaxLoadByMode(M)
            o_count++;
            Insert a.getPDRID() to pdrList;
        EndIf
        If a.getLoadByMode(M)>a.getMinLoadByMode(M)
```

```
          u_count++;
      EndIf
    EndFor
    If o_count==wfciList.size()
      AR(pdrList,M);        // Add new resource (See in Algorithm 3.2)
    EndIf
    If u_count==L.size()
      If !isUnderUtilized
        u_Time=CurrentTime;
        isUnderUtilized=true;
      Else
        If CurrentTime-u_Time>C
          RR(L,M);          // Add new resource (See in Algorithm 3.3)
        EndIf
      EndIf
    Else
      isUnderUtilized=false;
      u_Time=0;
    EndIf
  EndWhile
End
```

- Add new Resource

When all the existing PASE server have been overloaded, the PASE grid would automatically discover more computing resources and appropriately configure them to become newly available PASE servers to share the request workloads. Table 3-7 shows the structure of PASEProperty used to describe a PASE resource and its content stored in the PASE_Geninf in PIS. Table 3-8 describes the adding resource algorithm.

Table 3-7 Structure of PASEProperty

```
Structure PASEProperty {
        String     id;
        String     host;
        String     port;
        String     dbHost;
        String     dbPort;
        String     dbName;
        String     dbUser;
        String     dbPassword;
        String     pdrID;
        String     state
        // Different load determination mode have different value
        double    maxLoad;
}
```

Table 3-8 Adding Resource Algorithm

**Algorithm 3.2 (Adding Resource Algorithm)**

**Input:**

```
        The list of pdr's id L
        The load determination mode  M
AR(L,M):
01    Begin
02          // Get the list of AWs whose states are suspending
03          List sList=wfciPoolManager.getSuspendingPool();
```

```
    If sList.size()>0
        List temp=new List();
        For each AbstractWFCI aw∈ sList do
            If L.contains(aw.getPDR())
                Insert aw into temp;
            EndIf
        EndFor
        String pID = getMaxLoadByMode(temp);
        wfciPoolManager.moveSuspendingToRunning(pID);
        return;
    EndIf

    // Get the PDR to which the most overloaded PASE resources refer
    String pdrID=mostOccurence(L);
    List aList=pisManager.getReserving();
    For each PASEProperty p∈ aList do
        PASEProperty t=new PASEProperty();
        If p.maxLoad>t.maxLoad
            t=p;
        EndIf
    EndFor
    Remove t from aList;
    pisManager.updatePASEState(aList, "Ready");
    pisManager.replicatePDR(t.id,pdrID);
    wfciPoolManager.connectToServer(t.id);
End
```

- Remove Resource

  On the other hand, when the incoming requests decrease and the overall system has been under-utilized, the PASE grid will remove a portion of the PASE resources, allowing them to be utilized by other demanding PASE broker. Table 3-9 represents the removing resource algorithm. The algorithm just moves the AW representing the PASE resource to be removed removing into the suspending pool in WFCIPoolManager.

Table 3-9 Removing Resource Algorithm

```
Algorithm 3.3 (Removing Resource Algorithm)
Input:
        The list of running AWs L
        The load determination M
RR(L,M):
01  Begin
02      For each AbstractWFCI a∈wfciList do
03          AbstractWFCI t;
04          If a.getMaxLoadByMode(M)<t.getMaxLoadByMode(M)
05              t=p;
06          EndIf
07      EndFor
08      wfciPoolManager.moveToSuspending(t.getID());
09  End
```

In the WFCIPoolManager, the SuspendChecker periodically uses a suspending checking algorithm to check all the AWs in the suspending pool. For those AWs in which all workflow instances have finished, the SuspendChecker informs the GRTRManager to backup instances data and then asks the PISManager to clear the instance data and process definition data from the database of the PASE resource. Finally, the WFCIPoolManager disconnects the PASE resource from the PASE broker. Table 3-10 shows the suspending algorithm.

\

Table 3-10 Suspending Checking Algorithm

**Algorithm 3.4 (Suspending Checking Algorithm)**

**Input:**

```
    The pool of suspending AWs S
    The checking interval I
SC(S,I):
Begin
    List rList;
    While(true)
        Sleep(I);
        For each AbstractWFCI a∈S do
            If a.getInstance()==0
                Insert a into rList;
            EndIf
        EndFor
        For each AbstractWFCI aw∈rList do
            grtrManager.backup(aw.getID());
            pisManager.clearDB(aw.getID());
         wfciPoolManager.disconnectServer(aw.getID());
        EndFor
```

```
15          EndWhile
16      End
```

# Chapter 4. Performance Evaluation

Based on the PASE grid architecture described in Chapter 3, we have implemented a prototype system, and conducted a series of experiments for performance evaluation. Section 4.1 describes the configurations of the PASE grid environment and related experimental settings. A program that drives the series of experiments is described in Section 4.2. Finally, the results of experiments are shown and discussed in Section 4.3.

## 4.1 Experimental Settings

### 4.1.1 PASE Resources

In the following experiment, we include four PASE resources in the PASE grid environment. The information about the software and hardware configurations of each PASE resource is shown in Table 4-1. In addition, all PASE resources will use the same process definition repository in the experiment. The process definitions in the process definition repository are described in section 4.1.2

.

Table 4-1 PASE resources

| Resource Host | CPU | Memory | Database | Agentflow |
|---|---|---|---|---|
| 140.113.210.11 | AMD Athon64 1.81GHz | DDR II  1GB | MySQL 4.1 | 2.2.3.2 |
| 140.113.210.18 | AMD AthonXP 1.83GHz | DDR II  512 MB | MySQL 4.1 | 2.2.3.2 |
| 140.113.210.21 | AMD Athon64 1.81GHz | DDR II  1GB | MySQL 4.1 | 2.2.3.2 |
| 140.113.210.23 | AMD Athon64 1.81GHz | DDR II  1GB | MySQL 4.1 | 2.2.3.2 |

In the experiments, we explore three different types of metrics for defining the load limit on each PASE resource, The three types of metrics are *workflow instance number, request arrival rate*, and *average response time*. The first two metrics are workload directed, and the third is performance directed. Since the load limits should be directly related to user's awareness of system performance, the load limit values for the first two metrics are dependent on the computing capabilities of the underlying machines, and the load limit values for the third metric are consistent on all machines. The limit values used in the experiments are shown in Table 4-2. Since the memory space and the power of the CPU on 140.113.210.18 is smaller than on other machines, the limit values for the first two metrics on it is set to be lower than on others..

Table 4-2 Limits on three metrics of PASE resources

| Resource Host | Workflow instance number | Request arrival rate (per ms) | Average response time (ms) |
|---|---|---|---|
| 140.113.210.11 | 300 | 0.0005 | 2000 |
| 140.113.210.18 | 250 | 0.00025 | 2000 |
| 140.113.210.21 | 300 | 0.0005 | 2000 |
| 140.113.210.23 | 300 | 0.0005 | 2000 |

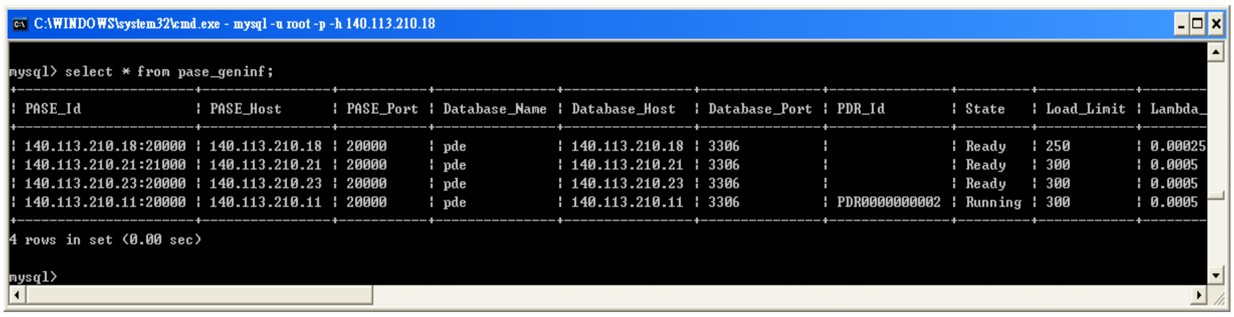
## 4.1.2 Process Definitions and PDR

The process definitions adopted in the experiments are real cases obtained from [16], which are used to construct a department management system in universities. The department management system includes several subsystems, such as 1) the working system for master students, 2) the working system for Ph.D. students, 3) bulletin system, 4) department computer & network center, and 5) laboratory. The services of these subsystems are defined with specific processes designed by and run on the Agentflow system.

In the following experiments, we created 1,500 member representing faculties, assistants and students, who manipulate department management system to accomplish all sorts of tasks which are present in daily operations of a department.

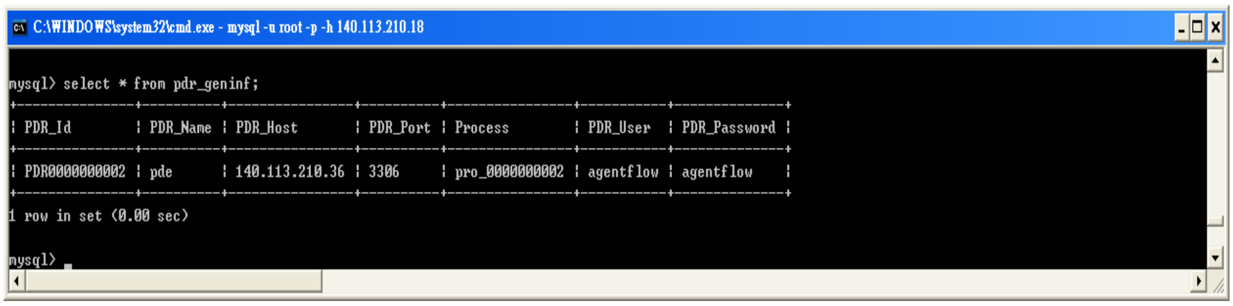
### 4.1.3 PASE Information Server

To establish a PASE grid, the PASE information server (PIS) first needs to set up several tables in the database server. These tables maintain the information about the PASE grid status, which has been described in Section 3.1.2. The following two figures show the essential data being stored into the database of PIS for the following experiment. In this experiment the PIS runs on 140.113.210.11:2099.



```
C:\WINDOWS\system32\cmd.exe - mysql -u root -p -h 140.113.210.18

mysql> select * from pase_geninf;
+----------------------+----------------+-----------+---------------+----------------+---------------+---------------+---------+------------+--------
| PASE_Id              | PASE_Host      | PASE_Port | Database_Name | Database_Host  | Database_Port | PDR_Id        | State   | Load_Limit | Lambda_
+----------------------+----------------+-----------+---------------+----------------+---------------+---------------+---------+------------+--------
| 140.113.210.18:20000 | 140.113.210.18 | 20000     | pde           | 140.113.210.18 | 3306          |               | Ready   | 250        | 0.00025
| 140.113.210.21:21000 | 140.113.210.21 | 20000     | pde           | 140.113.210.21 | 3306          |               | Ready   | 300        | 0.0005
| 140.113.210.23:20000 | 140.113.210.23 | 20000     | pde           | 140.113.210.23 | 3306          |               | Ready   | 300        | 0.0005
| 140.113.210.11:20000 | 140.113.210.11 | 20000     | pde           | 140.113.210.11 | 3306          | PDR0000000002 | Running | 300        | 0.0005
+----------------------+----------------+-----------+---------------+----------------+---------------+---------------+---------+------------+--------
4 rows in set (0.00 sec)

mysql>
```

Figure 4-1 PASE_Geninf

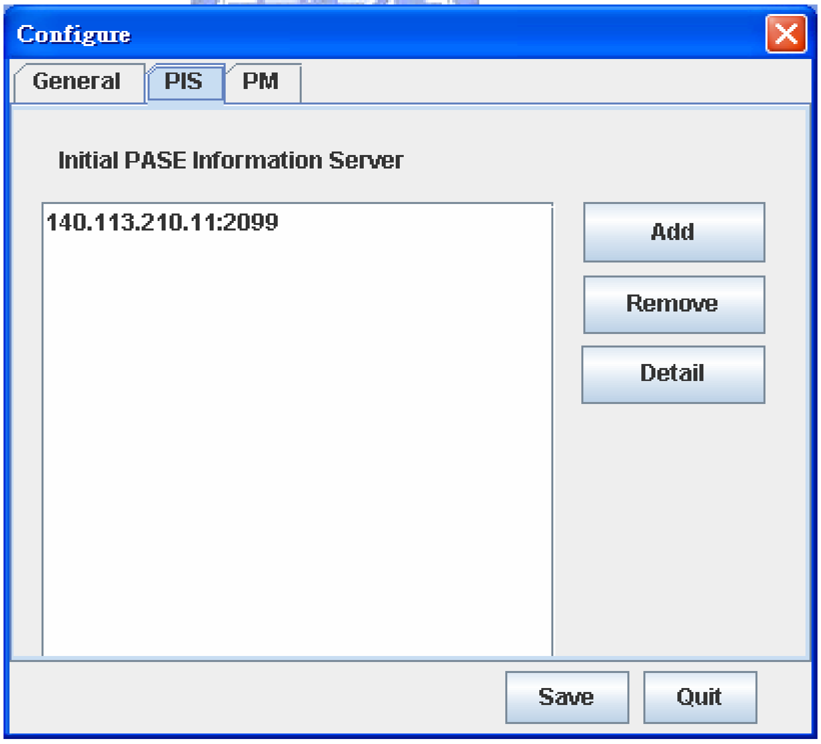


```
C:\WINDOWS\system32\cmd.exe - mysql -u root -p -h 140.113.210.18

mysql> select * from pdr_geninf;
+---------------+----------+---------------+----------+---------------+-----------+--------------+
| PDR_Id        | PDR_Name | PDR_Host      | PDR_Port | Process       | PDR_User  | PDR_Password |
+---------------+----------+---------------+----------+---------------+-----------+--------------+
| PDR0000000002 | pde      | 140.113.210.36 | 3306    | pro_0000000002 | agentflow | agentflow   |
+---------------+----------+---------------+----------+---------------+-----------+--------------+
1 row in set (0.00 sec)

mysql> _
```

Figure 4-2 PDR_Geninf

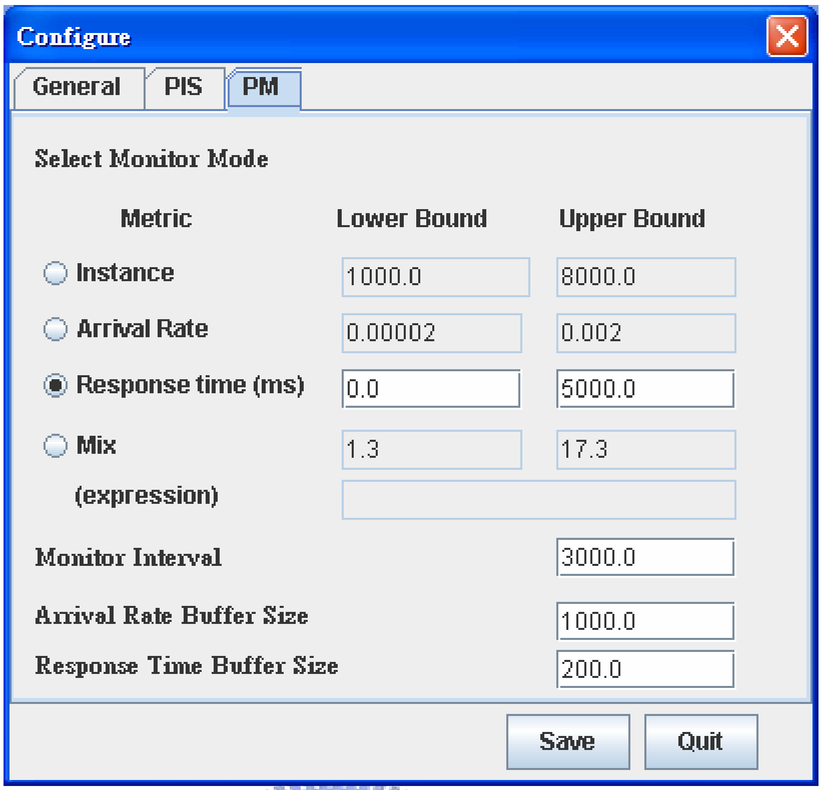
### 4.1.4 PASE Broker

Before the PASE broker start working, we must select some PISs and enter their host and port information into configure file of PASE broker. We also need to set the values of some attributes for the performance monitor. The snapshots of these two steps are shown in Figure 4-3 and Figure 4-4.



140.113.210.11:2099

Figure 4-3 PIS Configurations



Configure

General | PIS | PM

Select Monitor Mode

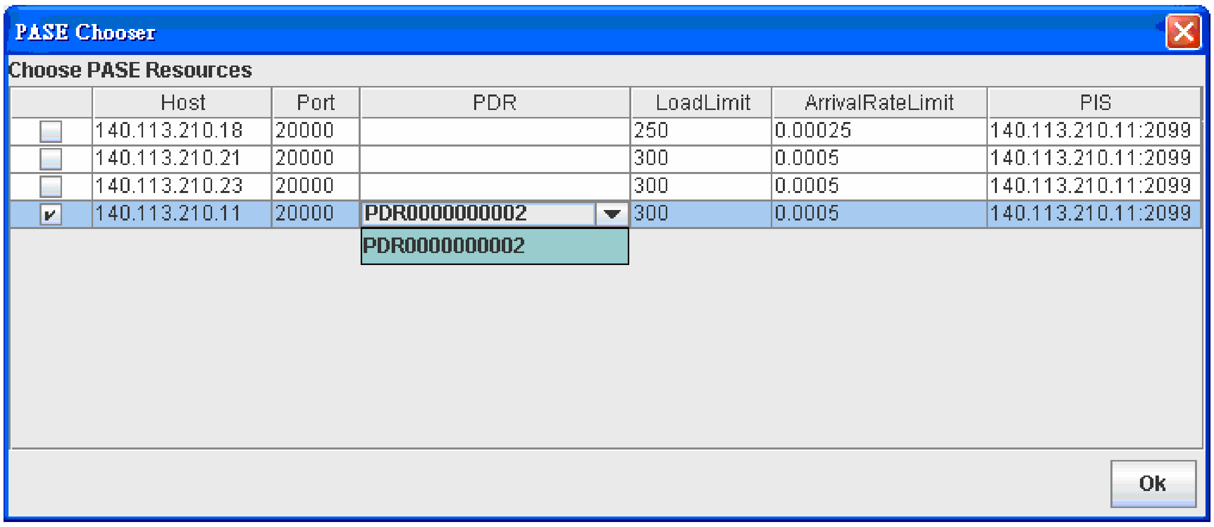
| Metric | Lower Bound | Upper Bound |
|---|---|---|
| Instance | 1000.0 | 8000.0 |
| Arrival Rate | 0.00002 | 0.002 |
| [✓ Response time (ms)] | 0.0 | 5000.0 |
| Mix | 1.3 | 17.3 |
| (expression) | | |

| | |
|---|---|
| Monitor Interval | 3000.0 |
| Arrival Rate Buffer Size | 1000.0 |
| Response Time Buffer Size | 200.0 |

Figure 4-4 Performance Monitor Configurations

In Figure 4-4, we can select a monitoring mode for the performance monitor, and set the lower bound as well as upper bound of that mode. The upper bound values in the Performance Monitor configurations are default values when the administrator does not set those values in the PIS. The Arrival Rate Buffer Size is the time interval for the PASE broker to measure the request arrival rate. The Response Time Buffer Size is the amounts of requests collected to measure the average response time.

When the above settings are completed, we can then start the PASE broker, and it will find some ready PASE resources can be used retrieved from PISs. In this experiment, at first we always add only one PASE resource and configure its corresponding PDR. Later on, if the incoming requests increasing and the system is overloaded, the PASE broker will automatically add a new PASE resource to the grid and configure its corresponding PDR. The snapshot of this step is show in Figure 4-5.


PASE Chooser

Choose PASE Resources

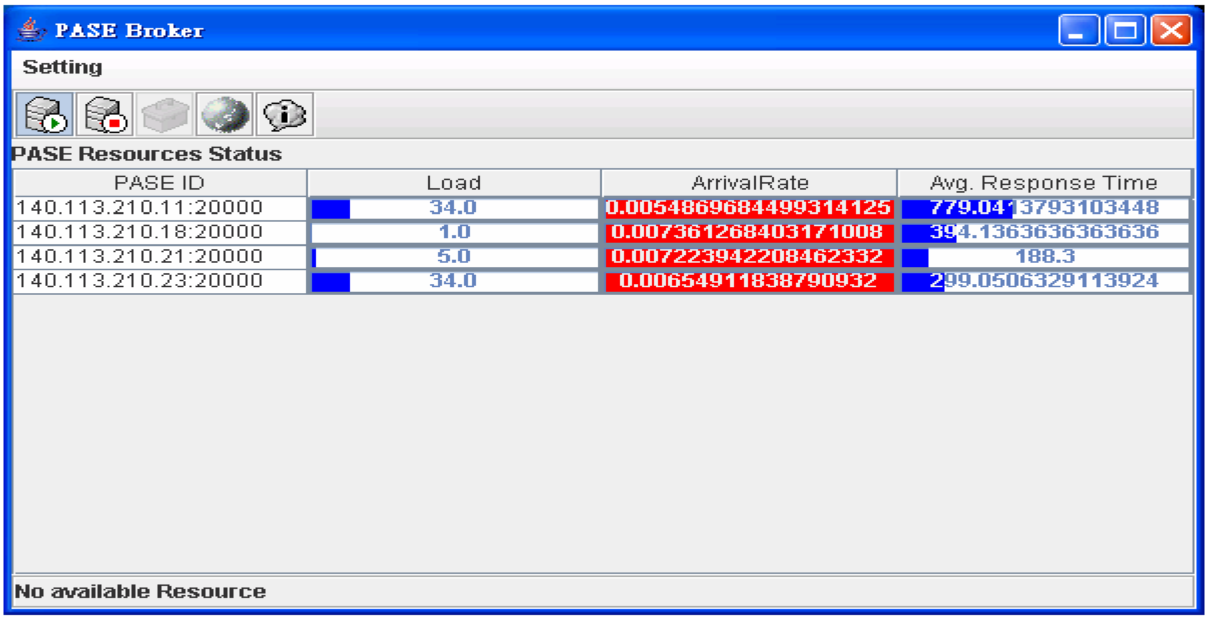
| | Host | Port | PDR | LoadLimit | ArrivalRateLimit | PIS |
|---|---|---|---|---|---|---|
| ☐ | 140.113.210.18 | 20000 | | 250 | 0.00025 | 140.113.210.11:2099 |
| ☐ | 140.113.210.21 | 20000 | | 300 | 0.0005 | 140.113.210.11:2099 |
| ☐ | 140.113.210.23 | 20000 | | 300 | 0.0005 | 140.113.210.11:2099 |
| ✓ | 140.113.210.11 | 20000 | PDR0000000002 | 300 | 0.0005 | 140.113.210.11:2099 |

PDR0000000002

Ok

Figure 4-5 Select initial PASE resource

Figure 4-6 is a snapshot of the runtime status of PASE resources in the PASE grid, The information shown includes the load (workflow instances number), average request arrival rate, and average request response times. For each PASE resource when one values exceed its upper bound set in the PIS, the representing progress bar will change its color will change their color from blue to red. When the overall system has been overloaded and there are no more available PASE resources, it will show the message ‘No available resource’ on the status bar of the PASE broker to inform administrators.


PASE Broker

Setting

PASE Resources Status

| PASE ID | Load | ArrivalRate | Avg. Response Time |
|---|---|---|---|
| 140.113.210.11:20000 | 34.0 | 0.0054869684499314125 | 779.0413793103448 |
| 140.113.210.18:20000 | 1.0 | 0.007361268403171008 | 394.1363636363636 |
| 140.113.210.21:20000 | 5.0 | 0.007223942208462332 | 188.3 |
| 140.113.210.23:20000 | 34.0 | 0.006549118387909320 | 299.0506329113924 |

No available Resource

Figure 4-6 System status

## 4.2 The Experiment Driver

A program for driving the following experiments was implemented. The main functions of the program are 1) generate requests to the PASE broker or to the single PASE server in original Agentflow architecture and 2) record the response time of each request and calculate the average response time. This program generates two kinds of random numbers for the experiments as follows:

- Arrival of requests

  The arrival rate of request is assumed to conform to the poisson distribution. In this experiment, the testing program generates four types of tasks, including createProcess(), startTask(), completeTask() and getTaskOfCompany().

- Task service time

  The task service time is assumed to conform to the exponential distribution just as in most queuing studies. Since in real workflow cases, most tasks usually involve human manipulation, such as filling out a form, the task service time here is used to simulate a user takes to do finish a task. The task service time is also equivalent to the time period between the startTask() request and the completeTask() request of a specific task.

Furthermore, the experiment driver program must simulate the behavior of the workflow engine, i.e. when a task is finished, then it needs to trigger the next task according to the corresponding process definitions. There should record finishing artifact state of each task in experiment driver program. Since each task could have multiple finishing states, the elimination of some states is required for prevention of infinite loop.

The experiment driver program requires the following six steps before starting an experiment:

(1) Select type of server, PASE Server or PASE broker

(2) Enter the host of server,

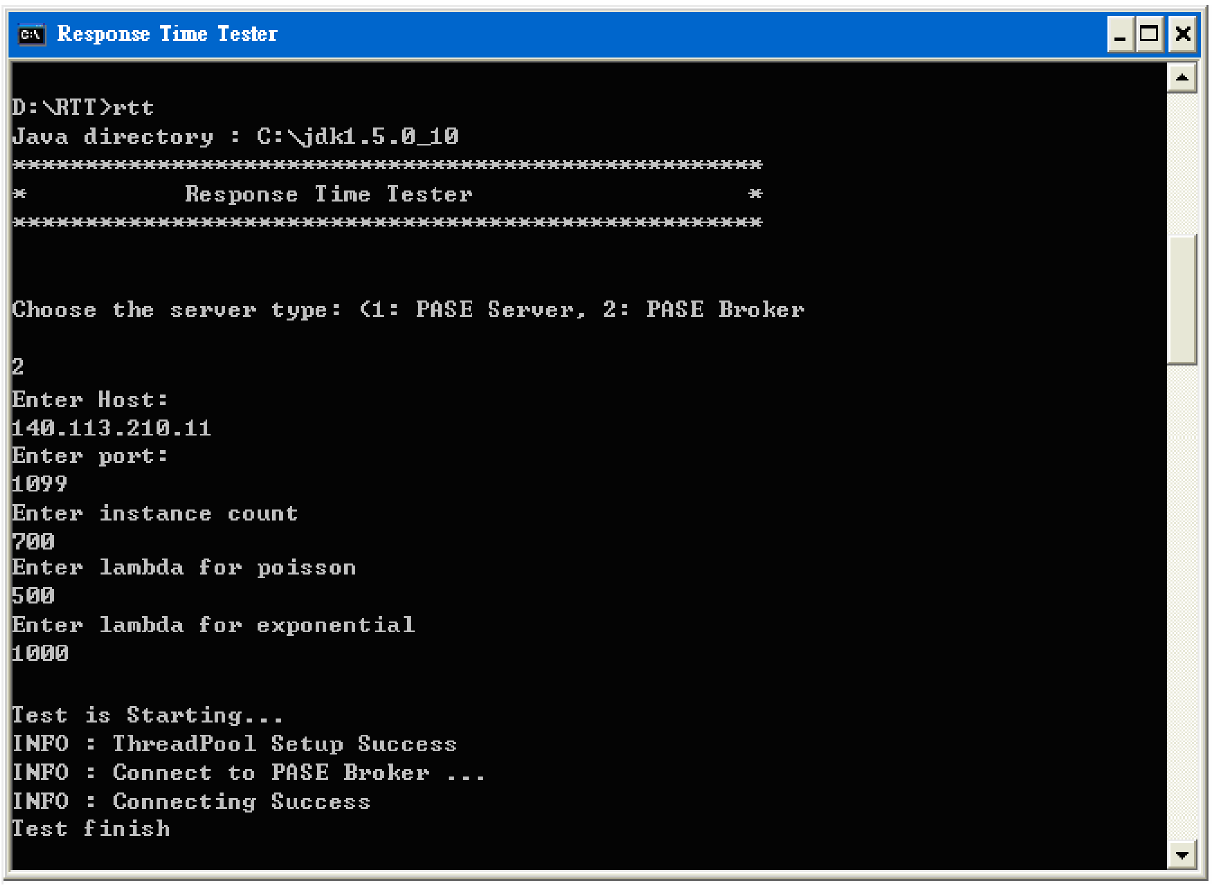
(3) Enter the port of server

(4) Enter the amounts of workflow instances to be generated

(5) Enter the lambda valuefor generating random number from the poisson distribution, and

(6) Enter the lambda value for generating random number from the exponential distribution.

Figure 4-7 illustrates these six steps.



```
D:\RTT>rtt
Java directory : C:\jdk1.5.0_10
****************************************************
*            Response Time Tester                  *
****************************************************


Choose the server type: (1: PASE Server, 2: PASE Broker

2
Enter Host:
140.113.210.11
Enter port:
1099
Enter instance count
700
Enter lambda for poisson
500
Enter lambda for exponential
1000

Test is Starting...
INFO : ThreadPool Setup Success
INFO : Connect to PASE Broker ...
INFO : Connecting Success
Test finish
```

Figure 4-7 Experiment Driver

## 4.3 Experiment Results

In the following experiment, the amounts of created workflow instances range from 50 to 2,500, the request average arrival rate is 0.002 requests/ms, and the average task service time is 1,000 ms. The requests considered in the experiments are createProcess(), startTask(), completeTask(), and getTaskOfCompany(). Four different experiments are conducted to evaluate the performance of four different architecture and mechanisms, including the single PASE server in the original Agentflow system and, the PASE grid architecture with three different load monitor modes, respectively. The following five figures illustrate the response time of the four kinds of requests in different workloads and under the four different system architectures, respectively.

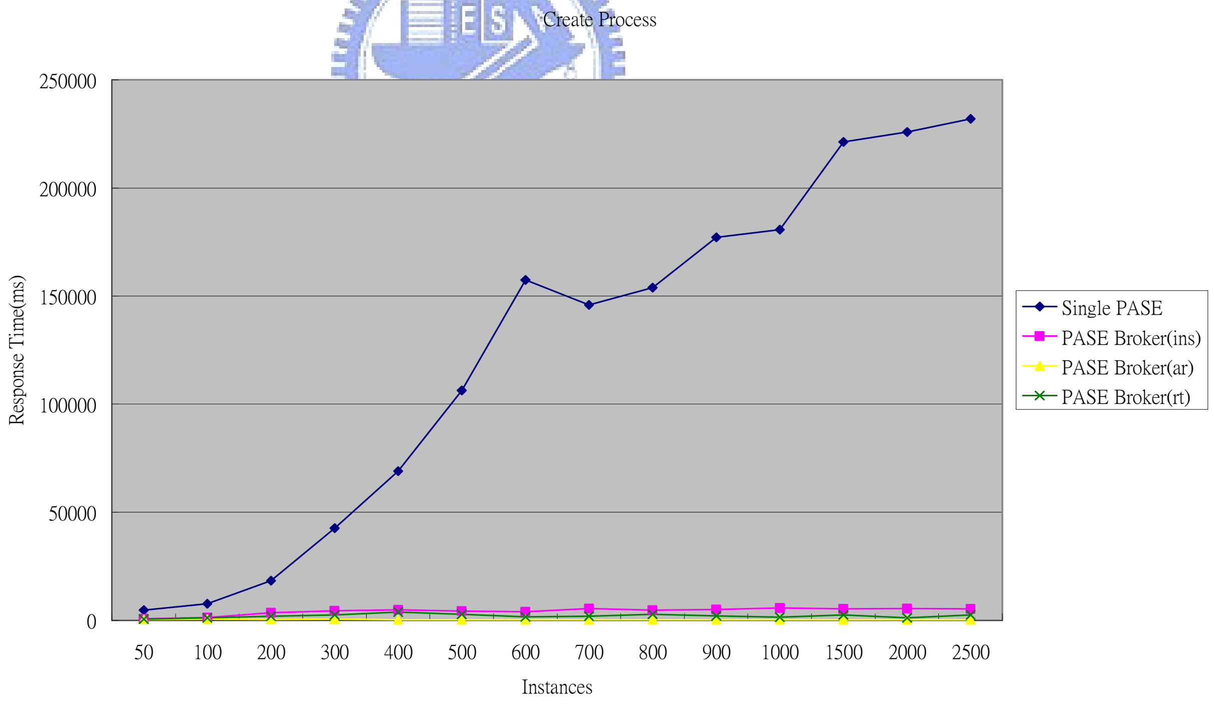


Figure 4-8 Results of createProcess()

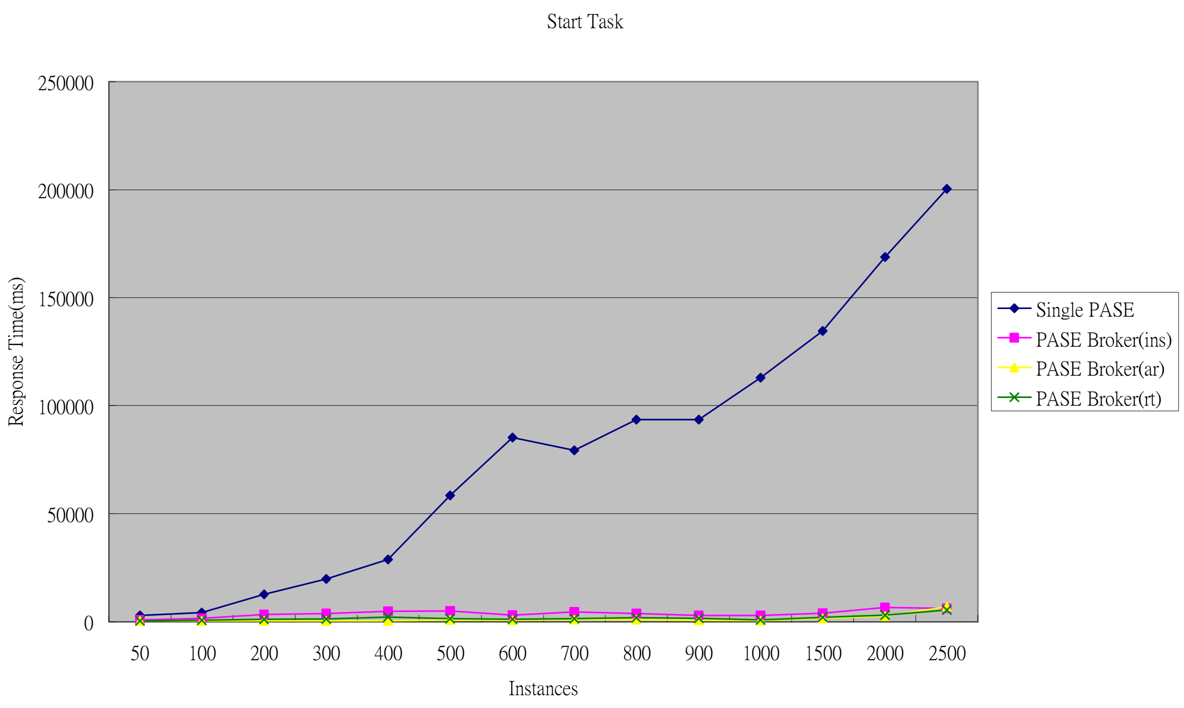


Figure 4-9 Results of startTask()

l

Figure 4-10 Results of completeTask()

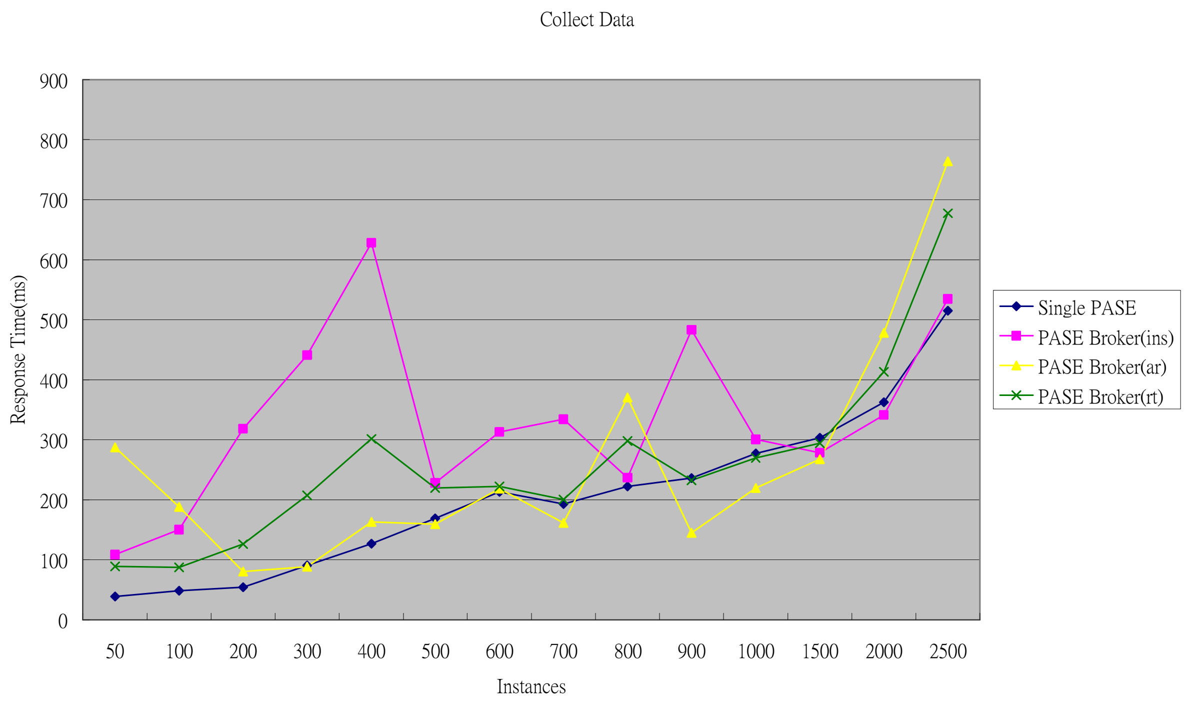


Figure 4-11 Results of getTaskOfCompany()

Obviously, the PASE grid architecture improves the runtime performance greatly compared to the original single-server Agentflow architecture for three of the four kinds of requests. Moreover, under different workloads, ranging from 50 to 2,500 instances, the PASE grid can deliver a nearly constant response time, benefiting from its scalable architecture. This is a desirable feature for moden service-oriented systems which have to confront unpredictable and dynamically changing amounts of incoming requests, while being expected to maintain acceptable and stable response time. The getTaskOfCompany() request is a special case, which must get the task instances from all running PASE resources. Therefore, in some situations, it may take even much longer time to finish than that in the original Agentflow architecture.

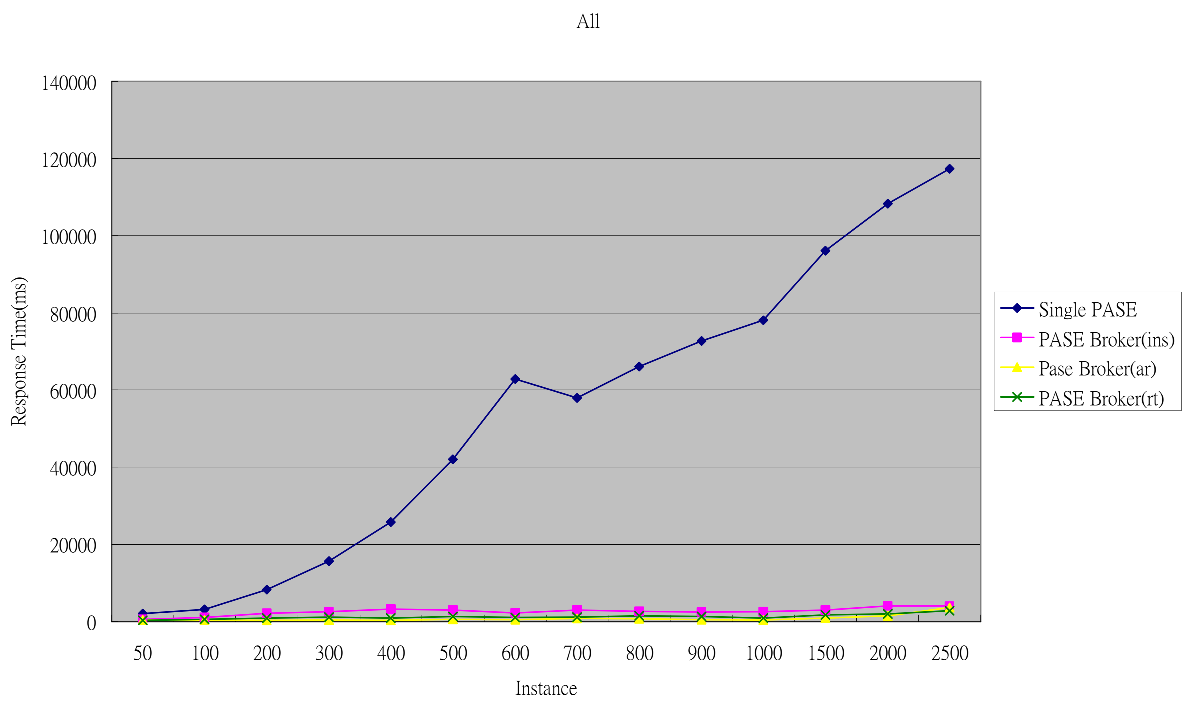


Figure 4-12 Average response time of all requests

Figure 4-12, shows the maximum average request response time of the single PASE server architecture is longer than 100,000 ms, while the maximum average request response time of the PASE grid architecture is shorter than 4,500 ms. This result includes that the PASE grid architecture proposed in this thesis, can effectively maintain an acceptable request response time under request loads of large variation..

The following article presents the comparisons among different monitoring modes of the PASE broker in detail. As seen the figures, the arrival rate mode and the response time mode in general outperform the instance mode. Moreover, the PASE broker with the arrival rate monitoring mode performs best and delivers a shorter and more stable average response time than with the other two monitoring mode. However, the performance of the arrival rate mode and the response time mode could be influenced by the corresponding buffer sizes set in the performance monitor. Therefore, relative performance of these two modes needs further studies, considering the effects of different sizes. This also raises an important issue that how to determine an appropriate buffer size becomes critical in delivering good and stable performance with these dynamic request dispatching algorithms.

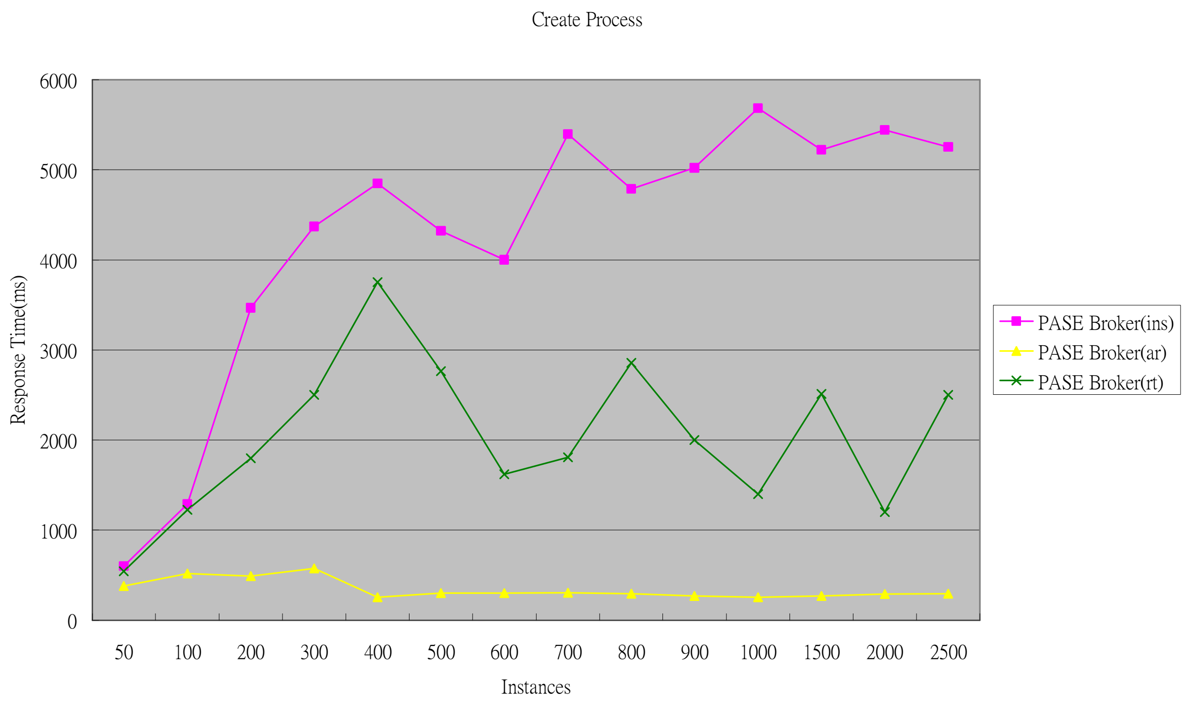


Figure 4-13 Performance results of createProcess()

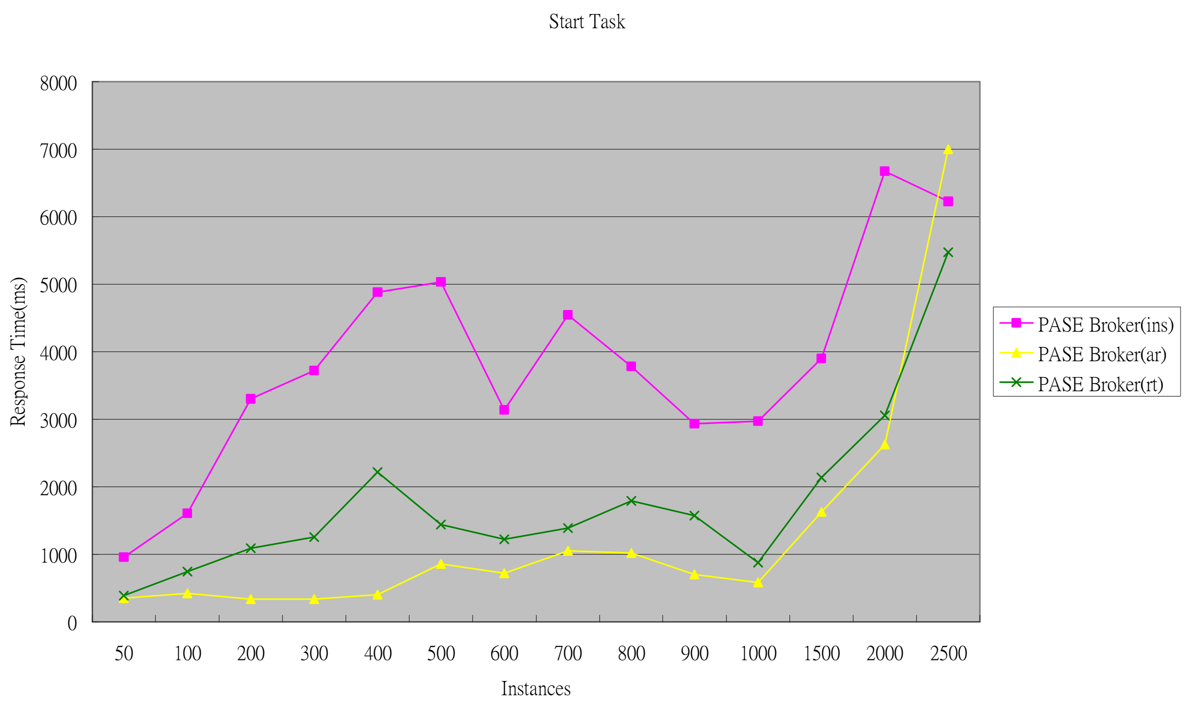


Figure 4-14 Performance results of startTask()

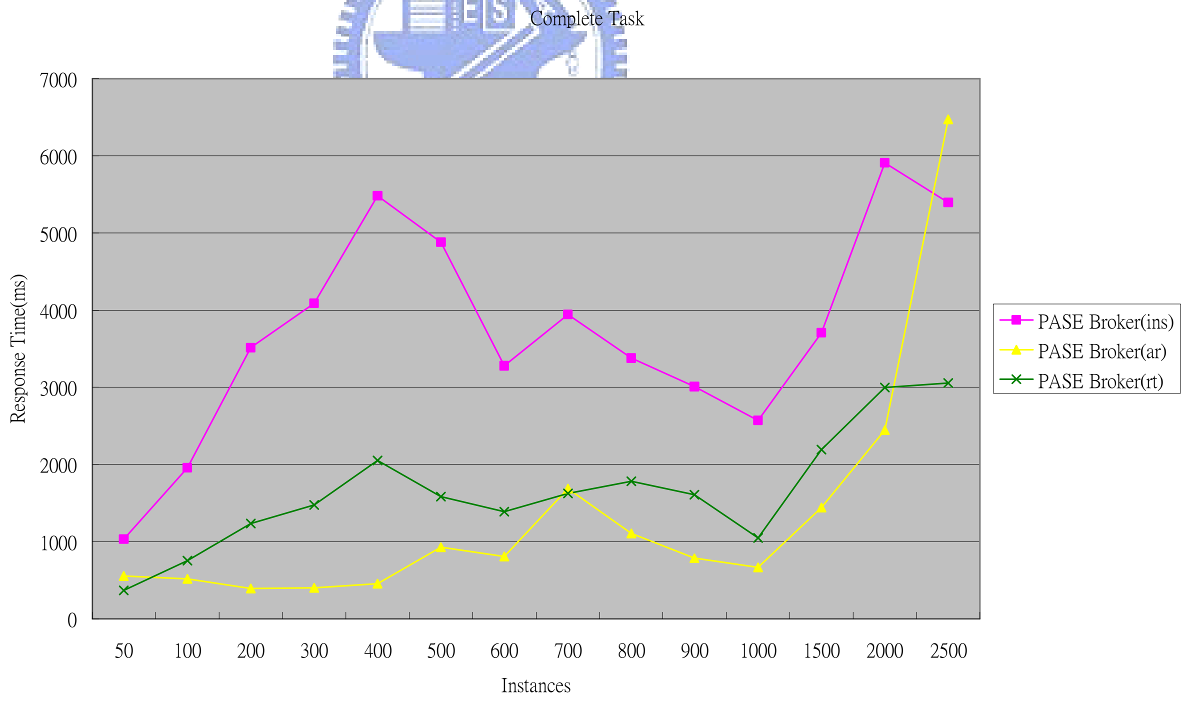


Figure 4-15 Performance results of completeTask()

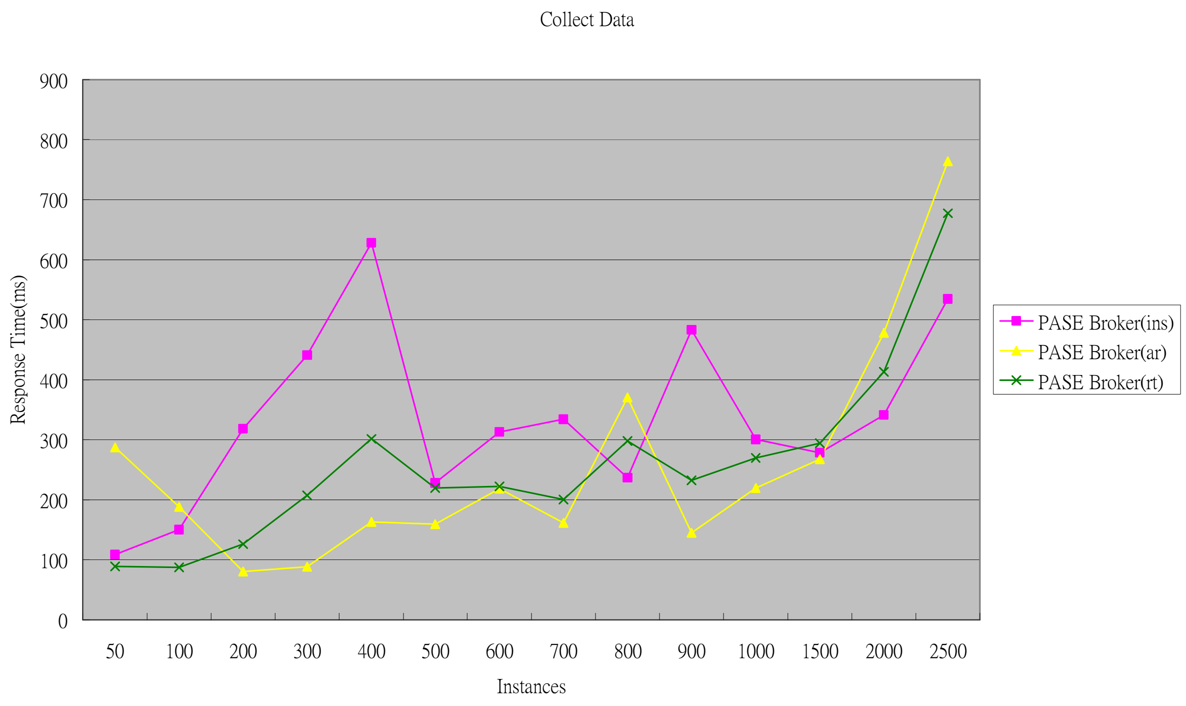


Figure 4-16 Performance results of getTaskOfCompany()

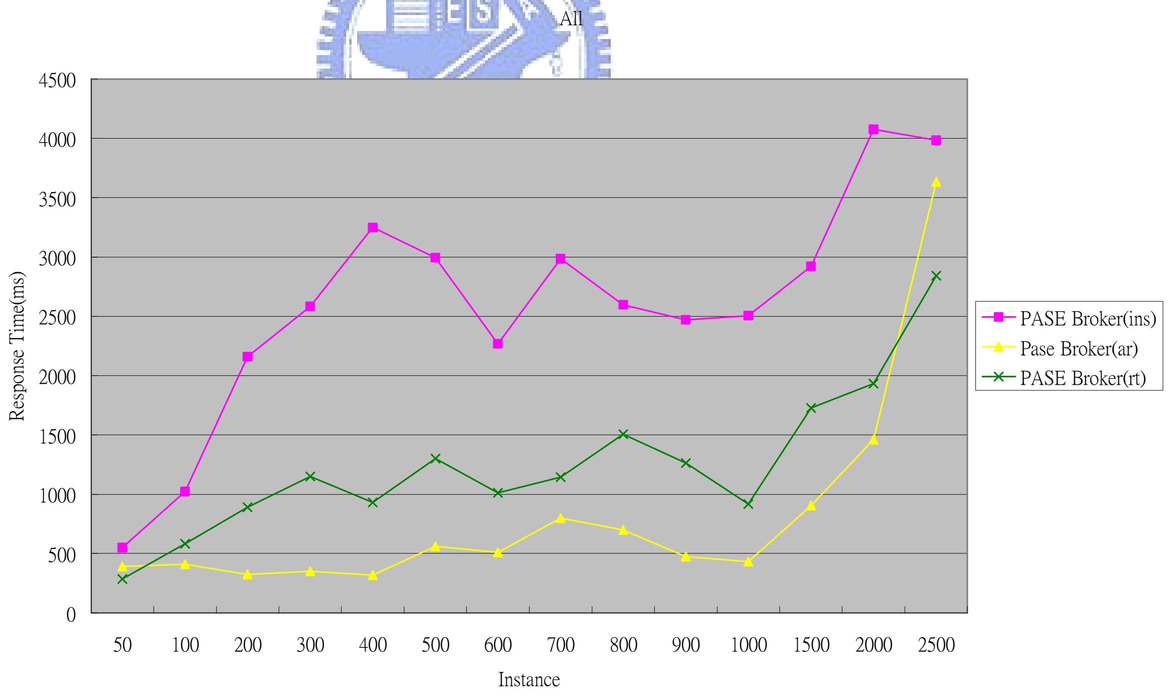


Figure 4-17 Average response time of all requests

# Chapter 5. Conclusions and future work

The major contribution of this thesis is propose a PASE grid architecture and, based on it, depelop a grid-enabled scalable workflow computing platform.. It achieves several on-demand resource provisioning and thus able to produce acceptable and stable request response time under a wide range of dynamically varying request workloads.  A prototype system has been implemented, and used to conduct a series of experiments for evaluating the performance of the proposed architecture.

The results of experiments shows that under different workloads, ranging from 50 to 2,500 workflow instances, the PASE grid can effectively deliver a nearly constant response time, benefiting from its scalable architecture. This is a desirable feature for moden service-oriented systems which have to confront unpredictable and dynamically changing amounts of incoming requests, while being expected to maintain a acceptable and stable response time

Based on our experience in the work and experiments described in this thesis, we point our some promising future research topics worth further investigations in the following.:

(1) Determining an appropriate buffer size for measuring average response time and request arrival rate is crucial for accurately representing the system workload. Further investigations are required on this issue in order to ensure that the dispatcher can effectively assign the income requests to appropriate PASE resources for delivering good and stable runtime performance.

(2) The Globus toolkit is currently the most widely deployed grid middleware. Integrating the services provided by Globus Toolkit into the PASE grid architecture could make our scalable workflow computing platform much easier to be adopted and implemented in current grid environment..

(3) Using history records to help predict future incoming requests is a promising approach to enable the dispatcher for making more appropriate allocation decisions

(4) Security is always one critical design issue in developing any grid applications and environments, so is it in our PASE grid architecture. This will be an important aspect in future extension of the PASE grid architecture.

# References

[1] Foster I, Kesselman C and Tuerke S. “The Grid: Blueprint for a New computing Infrastructure”, Morgan Kaufmann (2003).

[2] Foster I, “The Grid: A New Infrastructure for 21st Century Science”,.Physics Today, 55 (2). 42-47. 2002.

[3] D. De Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt. “The Evolution of the Grid”. In Grid Computing: Making The Global Infrastructure a Reality, pages 65–100. JohnWiley& Sons, 2003.

[4] Flowring Technology Corp, Agentflow system, http://www.flowring.com

[5] Catlett C, Smart L, “Metacomputing”, Communication of the ACM, vol. 35, no. 6, 1992

[6] FAFNER, http://www.npac.syr.edu/factoring.html

[7] Foster I, Geisler J, Nickless W, Smith W, and Tuecke S, “Software infrastructure for the I-WAY high performance distributed computing environment”, Proceedings of 5th IEEE Symposium on High Performance Distributed Computing, 1997.

[8] Kephart J.O , Chess D.M, “The vision of autonomic computing”, Computer, Vol:36, Issue: 1, 2003

[9] Laszewski Gv, Amin K, Hategan M, Zaluzec NJ, Hampton S, Rossi A, “GridAnt: A client-controllable Grid workflow system.”, Proceedings of 37th Hawaii International Conference on System Science, 2004.

[10] Shields M, Taylor I, “Programming scientific and distributed workflow with Triana services”, Proceedings of Workflow in Grid Systems Workshop in GGF 10, 2004.

[11] Krishnan S, Bramley R, Gannon D, Govindaraju M, Alameda J, Alkire R, Drews T, Webb E, “The XCAT science portal”, Proceedings of Supercomputing, 2001.

[12] Cao J, Jarvis SA, Saini S, Nudd GR, “GridFlow: Workflow management for Grid computing”, Proceedings of 3rd International Symposium on Cluster Computing and the

Grid, 2003.

[13] Altintas I, Berkley C, Jaeger E, Jones M, Ludaescher B, Mock S, “Kepler: Towards a Grid-enabled system for scientific workflows”, Proceedings of Workflow in Grid systems Workshop in GGF10, 2004.

[14] Czajkowski K, Fitzgerald S, Foster I, Kesselman C. “Grid Information Services for Distributed Resource Sharing”. Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing, 2001.

[15] Baru C, Moore R, Rajasekar A, Wan M, “The SDSC Storage Resource Broker”, Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research

[16] Shin-Jin Chou, Feng-Jian Wang, “Constructing a Management System for a University Department”, Master Thesis, National Chiao-Tung University, 2001.

[17] Mowshowitz A, “Virtual Organization”, Communication of the ACM, Vol.40, No. 9 , 1997

[18] The Globus Alliance, Globus Toolkit, http://www.globus.org

[19] Grimshaw A, Wulf W. et al., “The Legion Vision of a Worldwide Virtual Computer”, Communications of the ACM, vol.40, 1997.