The Journal of Systems and Software 83 (2010) 1538–1552

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A distributed server architecture supporting dynamic resource provisioning for BPM-oriented workflow management systems

Ching-Hong Tsai [a], Kuo-Chan Huang [b,*], Feng-Jian Wang [a], Chun-Hao Chen [a]

[a] Department of Computer Science, National Chiao-Tung University, No. 1001, Ta-Hsueh Road, Hsinchu, Taiwan
[b] Department of Computer and Information Science, National Taichung University, No. 140, Min-Shen Road, Taichung, Taiwan

ARTICLE INFO

Article history:
Received 6 March 2008
Received in revised form 15 April 2009
Accepted 3 April 2010
Available online 13 April 2010

Keywords:
Workflow management system
Business process management
Dynamic resource provisioning

ABSTRACT

Workflow management systems have been widely used in many business process management (BPM) applications. There are also a lot of companies offering commercial software solutions for BPM. However, most of them adopt a simple client/server architecture with one single centralized workflow-management server only. As the number of incoming workflow requests increases, the single workflow-management server might become the performance bottleneck, leading to unacceptable response time. Development of parallel servers might be a possible solution. However, a parallel server architecture with a fixed-number of servers cannot efficiently utilize computing resources under time-varying system workloads. This paper presents a distributed workflow-management server architecture which adopts dynamic resource provisioning mechanisms to deal with the probable performance bottleneck. We implemented a prototype system of the proposed architecture based on a commercial workflow management system, Agentflow. A series of experiments were conducted on the prototype system for performance evaluation. The experimental results indicate that the proposed architecture can deliver scalable performance and effectively maintain stable request response time under a wide range of incoming workflow request workloads.

1. Introduction

Workflow behavior appears in many different application domains. Various aspects of workflow computing research have received much attention in recent years. The work in (Yoo, 2009) deals with scheduling workflows in real-time applications. Duan et al. (2009) investigate the behavior and logical structure between tasks in a workflow, from running logs, to develop a better schedule for future running. Some researches developed methods for analyzing resource conflicts and correctness of workflow specifications (Hsu and Wang, 2008; Zeng et al., 2008).

To fulfill the ever growing needs of business process management (BPM) and automation, workflow management systems (WfMS) have been broadly adopted by many enterprises and organizations to (1) assign the required human resources and artifacts for executing each task, (2) control the business flows of tasks, and (3) monitor the executions of tasks, effectively. Standards, such as the Workflow Reference Model (Hollingsworth, 1995), WS-BPEL (2007), XPDL (2008), BPAF (2009), and BPMN (2009), have been defined by international organizations, e.g., WfMC (2009), OMG (2009), OASIS (2009), to facilitate the development of workflow and BPM applications.

Workflow management systems targeted at business process management (BPM) exhibit different features from those in the workflow management systems for scientific and engineering applications. Scientific workflow management systems, such as in (Deelman et al., 2005; Hoffa et al., 2008; Prodan, 2007; Prodan and Fahringer, 2008), usually deal with scheduling a set of inter-related tasks onto parallel or distributed computing resources, e.g., clusters (Buyya, 1999) or grids (Foster et al., 2003), where the relationships among the tasks can be represented as *directed acyclic graphs* (DAG) (Gerasoulis and Yang, 1993). Recently, many systems have been developed to support scientific workflows on grid platforms, including GridAnt (Laszewski et al., 2004), Triana (Shields and Taylor, 2004), XCAT (Krishnan et al., 2001), GridFlow (Cao et al., 2003), Kepler (Altintas et al., 2004), Pegasus (Deelman et al., 2005), ASKALON (Prodan, 2007), etc.

The tasks in scientific workflows usually require batch processing and take long times. On the other hand, BPM-oriented workflows usually consist of interactive tasks requiring less CPU-processing time, compared to those in scientific workflows, although they can also be represented by the DAG model. The interactive tasks in BPM-oriented workflows are involved with human interaction. Therefore, unlike tasks in scientific workflows, in

* Corresponding author. Tel.: +886 4 22183282; fax: +886 4 22183270.
E-mail addresses: chtsai@cs.nctu.edu.tw (C.-H. Tsai), kchuang@mail.ntcu.edu.tw (K.-C. Huang), fjwang@cs.nctu.edu.tw (F.-J. Wang).

0164-1212/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.04.001

addition to server-side processing time the total processing time of each task in BPM-oriented workflows also include thinking time or computer-manipulation time taken at the client side. The client-side thinking and computer-manipulation time is usually longer than the required processing time at the server side. In this way, the computational behavior in BPM-oriented workflows is more similar to the behaviors in many web-based applications than in scientific workflows.

Many researches tried to develop architectures or mechanisms for building high-performance web servers and web application servers (Wei and Xu, 2006; Chiang et al., 2008; Mendonça et al., 2008; Ranjan and Knightly, 2008; Urgaonkar et al., 2008; Wang et al., 2008; Mathur and Apte, 2009). Some of them specifically deal with multi-tier web application servers (Urgaonkar et al., 2008; Wang et al., 2008) or focus on QoS issues in web servers (Wei and Xu, 2006). However, most web-based applications, e.g., web servers providing static contents, can view different incoming requests as independent ones. Even for the requests coming from the same user, there are usually no prescribed relationships among them. Therefore, the server has no prior knowledge regarding the amount or arrival times of future incoming requests. The situation is different in BPM-oriented workflow management systems. Once a workflow is started, the server can expect how many task-execution requests will finally be generated in the future, although it has no idea about the exact arrival time of each future request.

This paper presents a distributed server architecture for BPM-oriented workflow management systems. Most existing BPM-oriented workflow management systems work based on simple client-server architecture with one single globally shared workflow engine and other tools such as database system to support the development and running of workflow applications. For example, *Agentflow*, a well-known JAVA-based WfMS developed by our laboratory and then Flowring Co. (Agentflow, 2008) in Taiwan, works with this structure. Obviously, the response time under such architecture is greatly affected by the computing power of the single centralized server. The increment of response time might not be tolerable when there are too many requests sent to the server within a short time period, i.e., the single centralized server becomes the performance bottleneck. Parallel server architecture can be used to alleviate the performance bottleneck. However, a static parallel architecture with a fixed-number of servers might not be efficient in resource utilization because the number of users and the requests generated by the users change from time to time in a in a real world WfMS. To deal with the issue, the distributed server architecture proposed in this paper provides scalable processing power with dynamic resource provisioning mechanisms, where the number of servers used is dynamically adapted to the time-varying incoming request workload.

This paper focuses on the aspect of dynamic resource provisioning in the proposed server architecture. A prototype system was implemented on a set of desktop PC's in our laboratory for performance evaluation. However, the proposed server architecture can be implemented in other distributed computing platforms, such as Clusters (Buyya, 1999) and Grids (Baru et al., 1998; Foster et al., 2003; Foster, 2002; Roure et al., 2003; Globus, 2008). In addition to the dynamic resource provisioning mechanisms discussed in this paper, implementations on different kinds of platforms may have to deal with other platform-dependent issues, such as communication costs, heterogeneity, reliability, etc.

The remainder of this paper is organized as follows. Section 2 introduces the system architecture of Agentflow. Section 3 presents the scalable workflow computing platform and describes several algorithms supporting the goal of dynamic resource provisioning collaboratively. A series of experiments for evaluating the performance of the proposed server architecture are presented in Section 4. Section 5 makes concluding remarks and points out some potential research works in the future.

2. A BPM-oriented workflow management system

This section introduces Agentflow (2008), a typical BPM-oriented WfMS. The Agentflow system (Agentflow, 2008) is a JAVA-based WfMS with centralized client-server architecture. There are three main components in Agentflow:

- Process definition environment (PDE). This is a graphical editor for modeling various views of a business, including process view, artifact view and organization view. Different views are modeled by separate tools in PDE, e.g., 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). This is a workflow enactment environment, which drives the flow of works and is also responsible for process enacting, control, management, and monitoring. The name 'PASE' stands for *Process Aided Software Engineering Environment*, which comes from an earlier version of workflow management system developed in our laboratory, targeted at software process management applications.
- Agenda. This is a client-side tool. Users can use it to browse their own task lists, do the tasks assigned to them, initiate processes, and monitor the states of the flow.
- An overview of the structure and relationships among these main components is shown in Fig. 1.

The database system inside Agentflow contains two repositories, *process definition repository* (PDR) and *runtime repository*. The process definition repository stores process definitions and the runtime repository keeps all instance data during workflow execution. Agentflow provides a JAVA-based application programming interface, Workflow Common Interface (WFCI), which allows direct interactions with the PASE server. For example, WebAgenda is a web-based agenda which communicates with the PASE server through the WFCI.

3. Scalable workflow management system with dynamic resource provisioning

This section extends Agentflow to a scalable workflow management system with dynamic resource provisioning. The scalable

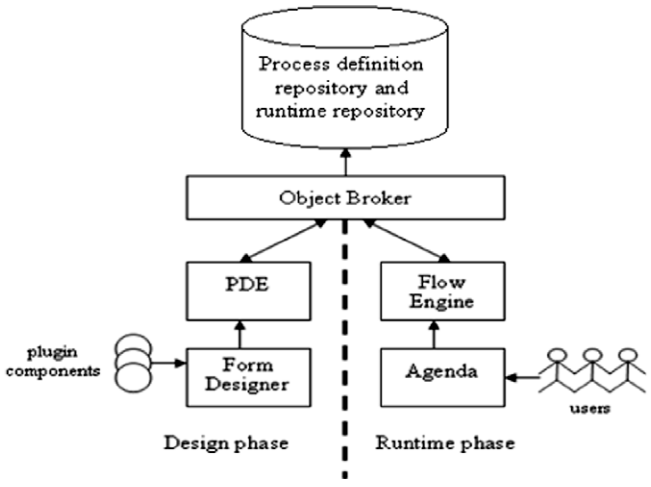

Fig. 1. Agentflow system overview.

system produces an acceptable and stable response time for requests under a wide range of request workloads. Here, we first present the system architecture and then the strategies for achieving on-demand resource provisioning.

3.1. Distributed server architecture

This section presents the distributed server architecture which equips Agentflow with dynamic resource provisioning capability. The distributed server architecture consists of one front-end server, called PASE broker, and several back-end servers running Agentflow, which are called PASE resources. Clients interact with the distributed workflow management system through the PASE broker, which provides a JAVA-based interface, PASE broker common interface (PBCI). Clients need not know the underlying server configurations such as the amount of PASE resources available and the computing speed of constituent machines. Therefore, the entire distributed workflow management system looks like a powerful and scalable super-Agentflow system. The distributed server architecture is shown in Fig. 2 and its constituent components will be elaborated in the following.

3.1.1. PASE resource

A PASE resource contains both hardware and software resources. The hardware resources are typically computers like PCs, notebooks, or workstations on which the software resources can run. The software resources include PASE servers and databases used to store runtime data and replicas of process definitions. The PASE server and database of a PASE resource can run on the same computer or on different machines. Each PASE resource is managed by one PASE information server (PIS), and it can be used by only one PASE broker at any instant.

3.1.2. Process definition repository and global runtime repository

Process definition repository (PDR) contains the business process definitions designed in process definition editor (PDE). When a PASE broker wants to add a new PASE resource, the PIS will replicate the corresponding content of PDR into the database of the PASE resource according to the incoming request. In each domain, there might be more than one PDR, and each PDR can be accessed by more than one PASE resource. Therefore, our approach allows the administrator to be in charge of the registration of PDR's into PIS's in addition. Each PDR itself can be a parallel or clustered database system which allows it to be fault-tolerant and consistent. Many modern database systems supporting the clustering feature can be used to implement PDR. For example, MySQL uses the MySQL Replication and MySQL cluster mechanisms to solve the issues of availability and scalability (MySQL, 2009).

Global runtime repository (GRTR) contains the workflow instances which are completely executed for future references. When a PASE broker wants to remove a PASE resource, it will first move the PASE resource's runtime data into GRTR. There is only one GRTR in the distributed server architecture, managed by the PASE broker. Although there is only one GRTR, GRTR itself can be a parallel or clustered database system to avoid the possible bottleneck issue in large systems. Like PDR, GRTR can be implemented by many modern database systems supporting the clustering feature, e.g., MySQL (2009). Such kind of database systems can be deployed to build the GRTR for large-scale workflow management systems.

3.1.3. PASE information server

PASE information server (PIS) plays a role similar to the MCAT in storage resource broker (Baru et al., 1998) or grid information service (GIS) in globus tookit (Czajkowski et al., 2001), maintaining necessary information about a domain, e.g., the information for all the PASE resources belong to the domain. Furthermore, it is responsible for replicating data from PDR into new PASE resources and clearing the database of removed PASE resources. The following tables describe the information maintained by PIS. The information is required for a PASE broker to discover, access, monitor, and manage PASE resources.

Table 1 illustrates the general information of a PASE resource. The unique id is formed by concatenating the host ID and port number (host:port). The state of a PASE resource can be *ready*, *reserved*, *running*, or *blocking*. A PASE resource is *ready* when the database is already created and the PASE server is initiated. The *reserved* state indicates that the PASE resource is reserved by some PASE broker, but not utilized by the PASE broker yet. The *running* state indicates that the PASE broker is using the PASE resource to

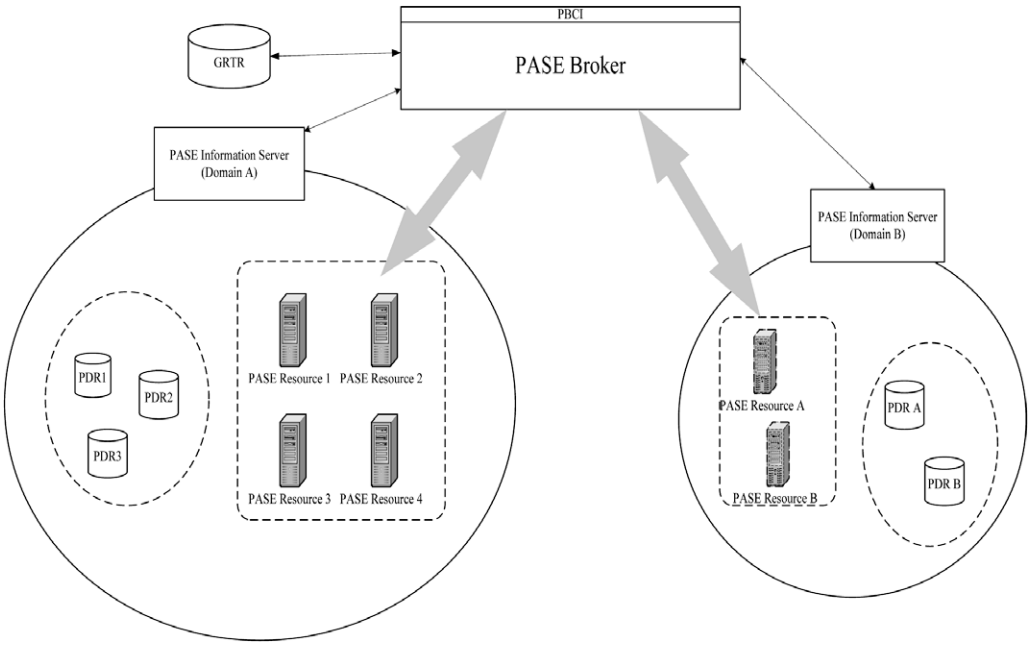

Fig. 2. The distributed server architecture.

Table 1
PASE general information.

Schedule	Description
PASE_ID	The unique id of a PASE resource
PASE_Host	The host of a PASE resource
PASE_Port	The port of a PASE resource
Database_Name	The database name of a PASE Resource
Database_Host	The host running the database of a PASE resource
Database_Port	The port of the database of a PASE resource
Database_User	The user's name for database access
Database_Password	The user's password for database access
PDR_Id	The id of the PDR to which a PASE resource has access
State	The state of a PASE resource
Load_Max_Limit	The upper limit on the load measured by the number of instances
Load_Min_Limit	The lower limit on the load measured by the number of instances
ArrivalRate_Max_Limi	The upper limit on the arrival rate
ArrivalRate_Min_Limit	The lower limit on the arrival rate

Table 2
Process definition repository general information.

Schedule	Description
PDR_ID	The unique id of a PDR
PDR_Name	The database name of a PDR
PDR_Host	The host of a PDR
PDR_Port	The port of a PDR
PDR_User	The user's name for PDR access
PDR_Password	The user's password for PDR access

serve incoming requests. The *blocking* state indicates the failure of a PASE resource.

Table 2 describes the general information and identification of a PDR. Each PDR has a distinct id which starts with "PDR". When an administrator registers a new PDR into the system, the PIS will generate this information according to the properties of the PDR.

3.1.4. PASE broker

A PASE broker coordinates PIS's, PASE resources, PDR's, and GRTR. The architecture of a PASE broker is illustrated in Fig. 3.

PIS Manager connects to and manages all PIS's. A PISC in Fig. 3 is a connection from the PASE broker to a PIS. PIS Manager periodically retrieves and caches the information maintained in PIS's. Initially, the administrator can select the PASE resources and the PDR's he/she wants to use, then PIS Manager sends replication request to all PIS's for replicating process definitions into their PASE resources. *PDR Manager* connects to and manages all PDR's. A PDRC in Fig. 3 is a connection from the PASE broker to a PDR. All the clients' requests of getting the process definition related data are handled by PDR Manager. *GRTR Manager* backups the completed workflow instances inside the PASE resource which is to be removed by the PASE broker.

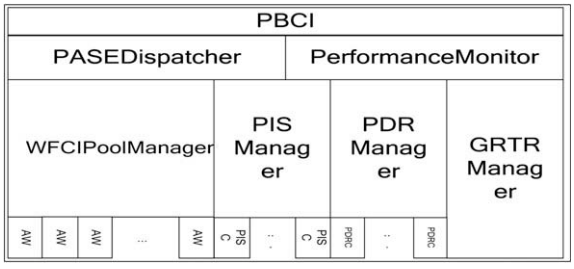

Fig. 3. The architecture of PASE broker.

WFCIPool Manager creates *AbstractWFCI's* (AW's) and connects them to the corresponding PASE resources with the JAVA RMI mechanism. Each AW wraps a WFCI connection and records some metadata about the connection, such as a list of processes and a list of member records. In addition, AW is defined with three metrics below to measure the workload for the reference of job dispatching.

WFCIPool Manager manages three pools: *running pool*, *suspending pool*, and *blocking pool*, corresponding to AW's of different states. The running pool contains the AW's providing services currently. The suspending pool contains the AW's which would not take any new process enactment requests but are still handling some unfinished workflow instances already running on them. The blocking pool contains the AW's which are at some failure states founded by the PASE broker.

Performance Monitor (PM) monitors the performance of the overall system based on the one or more load metrics specified by the administrator. These metrics include the number of instances, the average request arrival rate, and the average request response time. When the system is overloaded, Performance Monitor will inform WFCIPool Manager to find out more usable PASE resources from the PIS's in use under the order defined in PIS Manager, and create the connections to them. If there are no new PASE resources found, WFCIPool Manager replies an alert to the administrator and a new PASE resource is added manually. Moreover, when the system has been under-utilized in a (pre-)fixed time period, it also informs WFCIPoolManager to remove some AW's.

When a client sends a *process enactment request* (PER) to PASEDispatcher, it will select an appropriate PASE resource to instantiate the corresponding workflow definition according to a dynamic request dispatching algorithm which will be described in detail later. For efficiency of data sharing, all the tasks in a workflow will be allocated to the same PASE resource where the workflow is instantiated. Therefore, clients can send their requests except PER's directly to the specific PASE resources according to the global ID's of the tasks they want to manipulate. This arrangement can greatly reduce the burden of PASEDispatcher.

The following describes how clients can determine the destination PASE resources of their *task manipulation* requests. When a process is instantiated or a task is created on a PASE resource, the PASE resource generates a local ID for the process instance or the task. The local ID is unique within the PASE resource. However, the tasks and process instances on different PASE resources might have the same value for their local ID's. Therefore, a global ID is required to provide the uniqueness within the entire system. The global ID is also used for revealing the information of the PASE resource address. The global ID is formed by appending the corresponding PASE resource address to the local ID. An example of the mapping of local ID's to global ID's is shown in Table 3.

Each PASE resource performs necessary conversions between local and global ID's when it sends or receives process or task related information. Therefore, based on the global ID of the task to be manipulated, a client can find and send out its request to the PASE resource.

Table 3
Mapping between local and global ID's.

Global ID	PASE resource address	Local ID
Tsk(140.113.210.11:20000)000000000001	140:113.210.11:20000	Tsk000000000001
Proc(140.113.210.21:20000)000012345678	140.113.210.21:20000	Proc000012345678

3.2. On-demand resource provisioning strategies

This section discusses the resource provisioning strategies used in the distributed server architecture. There are two kinds of demands, where each has its own corresponding strategies.

3.2.1. User request processing

The distributed server architecture alleviates the performance bottleneck of the centralized server with its scalable computing capabilities, and thus produces shorter response time for user requests. Among various kinds of requests, *task manipulation* requests (TMRs) and *process enactment requests* (PERs) can benefit from this distributed server architecture. On the other hand, the *data collection* requests (DCRs) would need a little bit longer time than those in the original centralized architecture. How each kind of requests is processed is illustrated in the following:

- PER is used to create a workflow instance according to a predefined process definition. When a PER occurs, PASEDispatcher selects a PASE resource for processing the request according to the following dynamic request dispatching algorithm. For each PASE resource in the running pool, PASEDispatcher computes the additional workload that it can still accommodate by subtracting its current workload from its most sustainable workload specified by the administrator. The workload can be measured by three different modes: the number of instances, the average request arrival rate, and the average request response time, as described in the previous section. The maximum workload that a PASE resource can sustain is also represented in all the three modes. The PASE resource which can sustain the largest additional workload is chosen to handle the incoming PER. For example, assume there are five PASE resources in the system and at a moment the workloads on them are 15, 18, 9, 12, and 16, respectively, measured by the number of running workflow instances. If the maximum workload allowed on each PASE resource is set to 20 by the administrator, the additional workloads that the five PASE resources can accept are then 5, 2, 11, 8, and 4, respectively. Therefore, a new PER at that moment will be dispatched to the third PASE resource whose workload is 9. The detailed algorithm works as follows:

```
Algorithm: Dynamic request dispatching for PER
Input:
  The user id U
  The process id P
  The load metric used M
Output:
  A candidate PASE resource R
PER_RP (U,P,M):
01  Begin
02    // Get the list of AbstractWFCI's which are running
03    List wfciList=wfciPoolManager.getRunnings();
04    AbstractWFCI t = new AbstractWFCI();
05    set t = the first element in wfciList;
06    For each AbstractWFCI a ∈ wfciList do
07      // Compare current workload of each PASE resource
08      If (a.getMaxLoadByMode(M)-a.getLoadByMode(M))>
09        (t.getMaxLoadByMode(M)-t.getLoadByMode(M))
10        t = a;
11      EndIf
12    EndFor
13    R = t.getID();
14  End
```

- DCR is used to retrieve the instance related data or process-definition related data. A DCR may require more than one PASE resource to collaboratively accomplish its request and these PASE resources are determined by the data to be retrieved. TMR is used to manipulate a task or a group of tasks. It is sent to the PASE resource where the corresponding process instance it belongs to is created.

3.2.2. User request processing

Performance Monitor monitors the performance of each PASE resource in the system. It sends an event to WFCIPool Manager for adding new PASE resources or withdrawing some existing PASE resources when the entire system is overloaded or under-utilized. Performance Monitor checks each PASE resource in the running pool periodically to see if its current workload is larger than the maximum or lower than the minimum workload threshold, where both maximum and minimum are specified by the administrator, shown in Fig. 4. If the workloads of all PASE resources in the running pool exceed the maximum threshold, the system is *overloaded*. On the other hand, if the workloads of all PASE resources in the running pool are lower than the minimum workload threshold for a pre-defined time period, the system is deemed as *under-utilized*.

For example, assume the maximum workload threshold and minimum workload threshold are set to 100 and 10, respectively in a system with five running PASE resources. Performance Monitor keeps monitoring the status of each PASE resource periodically at a fixed time period specified by the administrator, e.g., one second. If at some time instance t1 the workloads of the five PASE resources are 101, 110, 106, 108, and 115, respectively, all larger than the maximum workload threshold, 100, Performance Monitor will signal that the system is overloaded and try to add one new PASE resource into the running pool to share the incoming workload. On the other hand, if at some time instance t2 Performance Monitor finds that the workloads of the five PASE resources are 6, 8, 7, 3, and 4, respectively, all less than the minimum workload threshold, 10, it will set the system state to be under-utilized. From then on, each time Performance Monitor checks the status of PASE resources, if at least one of the under-utilized PASE resource has its workload raised over the minimum workload threshold, Performance Monitor sets the system state from under-utilized back to normal. Otherwise Performance Monitor calculates the duration that the system has been under-utilized. Once the under-utilized duration exceeds the predefined threshold, e.g., 10 s, Performance Monitor will notify the system to remove one PASE resource from the running pool for increasing the utilization of remaining PASE resources. In summary, to maintain an acceptable level of request

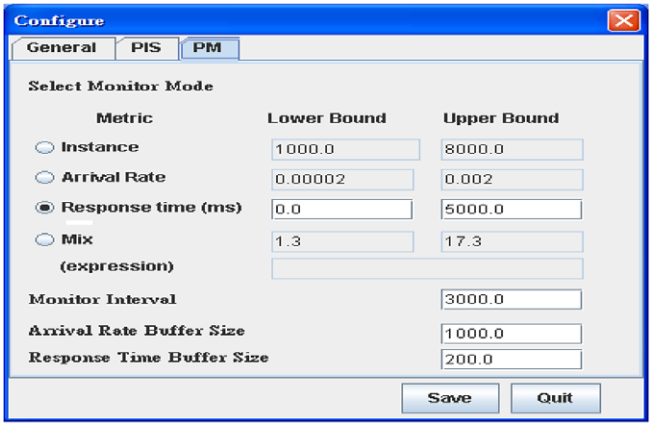

Fig. 4. Performance Monitor Configurations.

response time the system will immediately add one new PASE resource into the running pool once overloaded. On the other hand, the system will remove a PASE resource only after having remained under-utilized for a specific time duration, in case that the system returns to the normal state or even the overloaded state very soon after a very short time period of under-utilization. The detailed performance monitoring algorithm is described in the following:

```
Algorithm: Performance monitoring
Input:
  The monitoring interval I
  The list of running AWs L
  The load metric used M
A time period C
/* When the under-utilized time period exceeds this
  predefined threshold, the system will remove some PASE
  resources. */
PM  (I,M,C,L):
01  Begin
02    long u_Time = 0;
03    List pdrList;
04    boolean isUnderUtilized = false;
05    While(true) do
06      sleep(I);
07      int o_count = 0;
08      int u_count=0;
09      For each AbstractWFCI a ∈ L do
10        If a.getLoadByMode(M)>a.getMaxLoadByMode(M)
11          o_count++;
12          Insert a.getPDRID() to pdrList;
13        EndIf
14        If a.getLoadByMode(M)< a.getMinLoadByMode(M)
15          u_count++;
16        EndIf
17      EndFor
18      If o_count==L.size()
19        RA(pdrList,M); // Add new PASE resources
                // will be described in the
                // next section
20      EndIf
21      If u_count==L.size()
22        If !isUnderUtilized
23          u_Time=CurrentTime;
24          isUnderUtilized=true;
25        Else
26          If CurrentTime-u_Time>C
27            RW(L,M);  // Add new resource
//(See the Algorithm in Table 6)
28            EndIf
29        EndIf
30      Else
31        isUnderUtilized=false;
32      EndIf
33    EndWhile
34  End
```

Once all running PASE resources are overloaded, WFCIPool Manager might apply the following resource-addition algorithm to discover computing resources outside and set them as available PASE resources for use. Our resource addition is done gradually in order to reduce variation. In the algorithm, WFCIPool Manager firstly finds a set of PASE resources from the suspending pool whose corresponding PDR's are running and then moves the PASE resource with the largest workload among them to the running pool. For each run of PASE resource addition, WFCIPool Manager is designed to choose the PASE resource which increments the least computing power among the resources discovered, and puts it into the pool. The selection method tries to avoid the time overhead required for setting up a new PDR by choosing the candidate PASE resource from the suspending pool first.

For example, at the time the system is overloaded, assume there are five PASE resources in the suspending pool with the workloads, 9, 12, 18, 4, and 6, respectively. The system will pick up the third PASE resource with the workload value 18 to add it into the running pool for gradual computing power increase, since the PASE resource has the highest workload and thus can provide the least additional computing power to the system. On the other hand, if the suspending pool is empty when the system is overloaded, the system will try to find a new PASE resource with the least computing power to achieve the goal of gradual performance enhancement. The algorithm details are below:

```
Algorithm: Resource adding
Input:
  The list of PDR's L
  The load metric used M
RA(L,M):
01 Begin
02 // Get the list of AW's whose states are suspending.
  // Search for available resources from the suspending
  // PASE resources first.
03  List sList=wfciPoolManager.getSuspendingPool();
04  If sList.size()>0
05    String pID = getMaxLoadByMode(sList,M);
06    wfciPoolManager.moveSuspendingToRunning(pID);
07    return;
08  EndIf
09 // If no suspending PASE resources, search from PASE
  // resources in the reserved state.
10 // Get the least loaded PDR.
11  String pdrID=leastLoaded(L);
12  List aList=pisManager.getReserving();
13  PASEProperty t= the first element in aList;
14  For each PASEProperty p∈ aList do
15    If p.maxLoad < t.maxLoad
16      t = p;
17    EndIf
18  EndFor
19  Remove t from aList;
20  pisManager.updatePASEState(t, "Ready");
21  pisManager.replicatePDR(t.id,pdrID);
22  wfciPoolManager.connectToServer(t.id);
23 End
```

On the other hand, when the incoming requests decrease and the overall system has been under-utilized, the system will release a portion of the resources for use by other demanding applications. WFCIPool Manager uses a resource withdrawing algorithm to select a running PASE resource and move it to the suspending pool. Corresponding to the above addition algorithm, WFCIPool Manager follows a gradual-shrink policy, i.e., it withdraws the PASE resource with the least processing power in the running pool. In the system PASE resources with higher processing power will be configured with higher maximum workload threshold values. Therefore, in the algorithm, the system finds the PASE resource with the least processing power by selecting the PASE resource with the smallest maximum workload threshold. For example, if the running pool contains five PASE resources with the maximum workload threshold values, 100, 90, 85, 60, and 120, respectively. The fourth PASE

resource will be selected for removing since it has the least value of the maximum workload threshold. The detailed resource withdrawing algorithm works as follows:

```
Algorithm: Resource withdrawing
Input:
  The list of running AW's L
  The load metric used M
RW(L,M):
// remove the PASE resource with the least computing
// capability first
01  Begin
02  AbstractWFCI t= the first element in L;
03    For each AbstractWFCI a∈ L do;
04      If
  a.getMaxLoadByMode(M)<t.getMaxLoadByMode(M)
05        t = a;
06      EndIf
07    EndFor
08    wfciPoolManager.moveToSuspending(t.getID());
09  End
```

WFCIPool Manager periodically checks all the AW's in the suspending pool. For those AW's having finished all workflow instances on them, it first informs the GRTR Manager to backup instance data and then asks PIS Manager to clear up the instance data as well as the process definition data in the PASE resources' databases. Finally, WFCIPool Manager disconnects these PASE resources from the PASE broker. The following algorithm describes the procedure in detail.

```
Algorithm: Suspending pool monitoring algorithm
Input:
  The pool of suspending AWs S
  The checking interval I
SC(S,I):
01  Begin
02    List rList;
03    While(true)
04      Sleep(I);
05      For each AbstractWFCI a∈ S do
06        If a.getInstance()==0
07          Insert a into rList;
08        EndIf
09      EndFor
10      For each AbstractWFCI aw∈ rList do
11        grtrManager.backup(aw.getID());
12        pisManager.clearDB(aw.getID());
13      wfciPoolManager.disconnectServer(aw.getID());
14      EndFor
15    EndWhile
16  End
```

4. Performance evaluation

Based on the distributed server architecture described in Section 3, we have implemented a prototype system and conducted a series of experiments for performance evaluation.

4.1. Experimental setting

In the following experiments, we set up a distributed server system consisting of four PASE resources. The information about the software and hardware configurations of each PASE resource is shown in Table 4. All PASE resources will use the same process definition repository in the experiments.

Table 4
Configurations of PASE resources.

Resource Host	CPU	Memory	Database	Agentflow
140.113.210.11	AMD Athlon64 1.81GHz	DDR II 1GB	MySQL 4.1	v. 2.2.3.2
140.113.210.18	AMD AthlonXP 1.83GHz	DDR II 512 MB	MySQL 4.1	v. 2.2.3.2
140.113.210.21	AMD Athlon64 1.81GHz	DDR II 1GB	MySQL 4.1	v. 2.2.3.2
140.113.210.23	AMD Athlon64 1.81GHz	DDR II 1GB	MySQL 4.1	v. 2.2.3.2

Table 5
Load limit values of three different metrics.

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

In the experiments, we explore three different load metrics for defining the load limit on each PASE resource, including *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 limit 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 5. Since the memory space and CPU power on 140.113.210.18 are smaller than on other machines, the limit values for the first two metrics on it are set to be lower than on others.

The process definitions adopted in the experiments are real cases obtained from (Chou and Wang, 2001), which are used to construct a department management system in universities. The department management system includes five subsystems: (1) the working system for M.S. students, (2) the working system for Ph.D. students, (3) Bulletin system, (4) Department Computer and Network Center, and (5) Laboratories. The services provided by these subsystems are defined and run on Agentflow. In the following experiments, we created 1500 members representing faculties, assistants and students, who manipulate department management system to accomplish various sorts of tasks commonly seen in daily operations of a department.

Through the interface shown in Fig. 4, we can select a load metric for performance monitoring and set the lower bound as well as the upper bound. The upper bound values in the performance monitor configurations are the default values if the administrator does not set those values in 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 amount of requests collected to measure the average response time.

After finishing the above setting, we can then start the PASE broker and it will find some ready PASE resources from the PIS. In this experiment, at first we add only one PASE resource and configure its corresponding PDR. Later on, if the incoming requests increase and the system is overloaded, the PASE broker will automatically add a new PASE resource to the system and configure its corresponding PDR. A snapshot of this procedure is show in Fig. 5.

Fig. 6 is a snapshot of the runtime status of PASE resources in the system. The information shown includes the load values in three different metrics. For each PASE resource, when the load value exceeds its upper bound set in PIS, the progress bar will change its 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 the administrator. Figs. 7–10

A program for driving the experiments is implemented as follows. The main functions of the program are: (1) generate requests

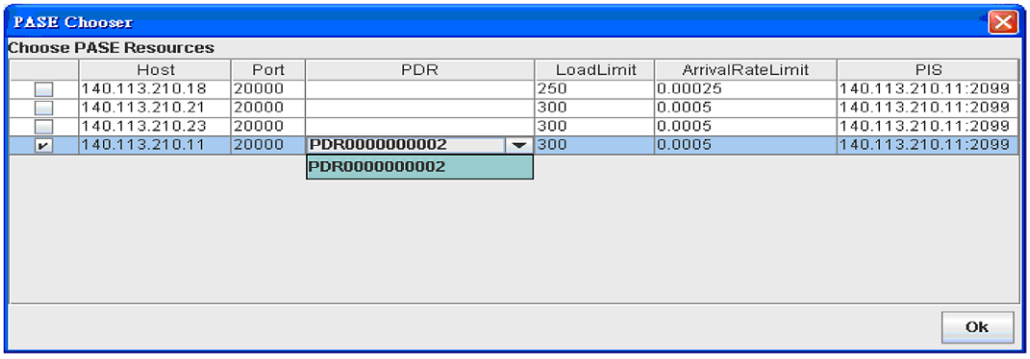

Fig. 5. Configure the initial PASE resource.

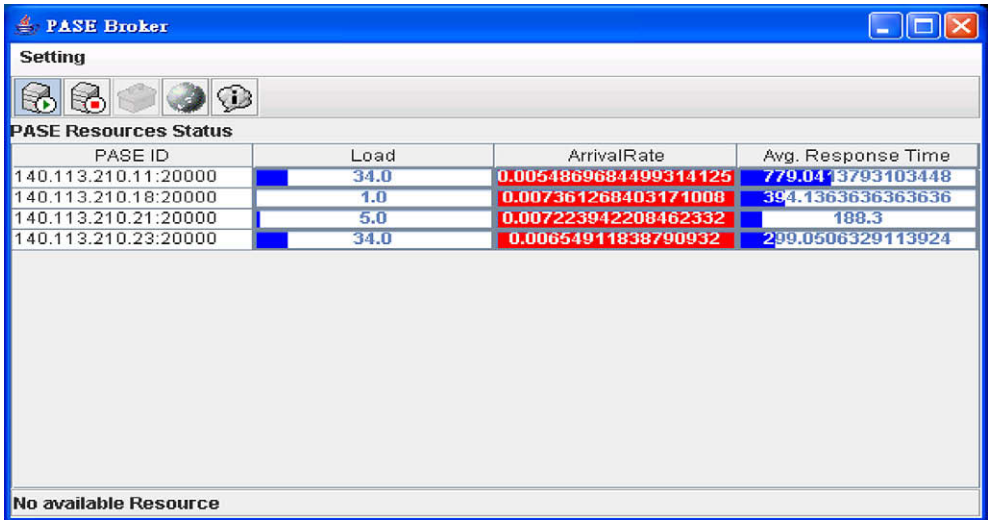

Fig. 6. System status.

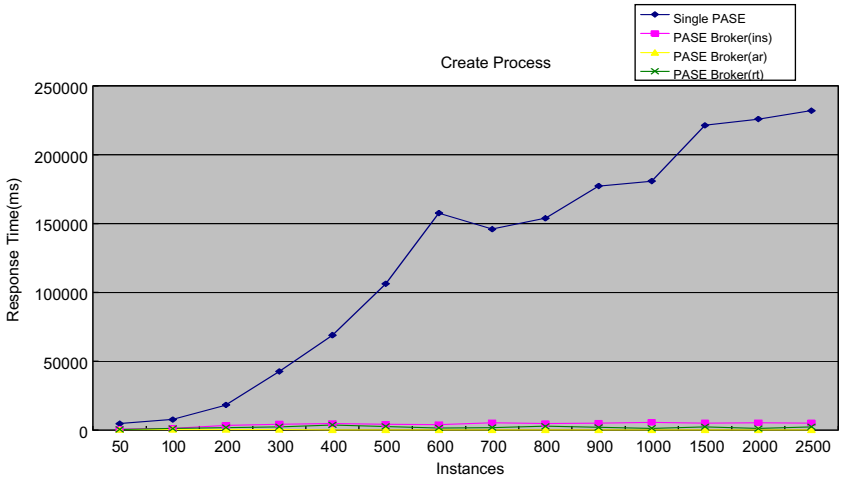

Fig. 7. Performance results for create Process().

to the PASE broker and PASE resources, or to the single PASE server in the 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 rate of requests

The arrival rate of requests is assumed to conform to the Poisson distribution. In the experiments, the test program generates four types of tasks, including create Process(), start Task(), complete Task() and get TaskOfCompany().

- 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 the time a user takes to finish a task. The task service time is equivalent to the time period between the startTask() and complete Task() requests of a specific task.

In the following experiments, the amounts of workflow instances range from 50 to 2500, the request arrival rate is 0.002 requests/ms, and the average task service time is 1000 ms. The requests considered in the experiments are createProcess(), start Task(), complete Task(), and get TaskOfCompany(). Four different experiments are conducted to evaluate the performances of four different scenarios, including a single PASE server in the original

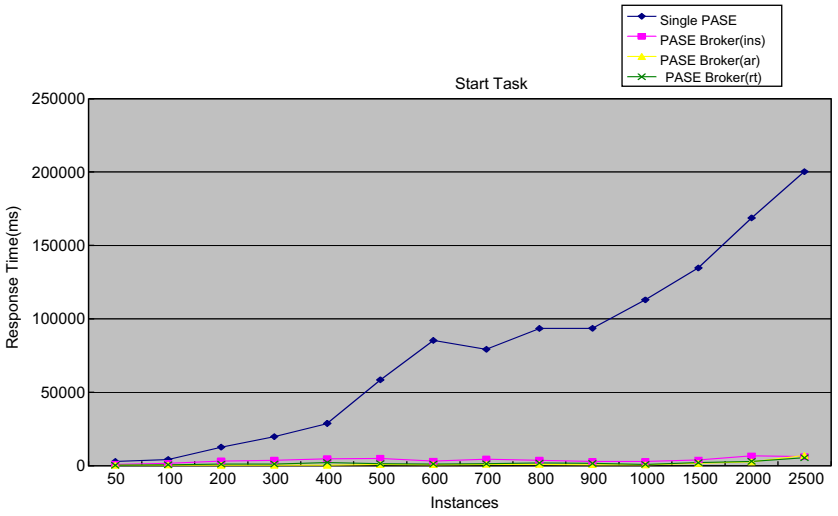

Fig. 8. Performance results of start Task().

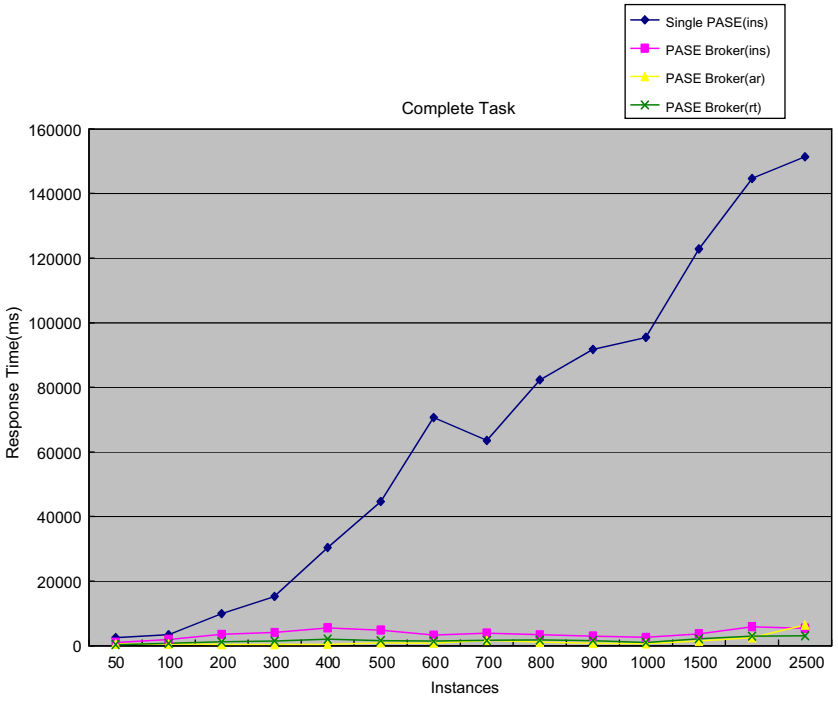

Fig. 9. Performance results of complete Task().

Agentflow architecture and the distributed server architecture with three different load metrics, respectively. The following figures illustrate the average response time of the four kinds of requests under different workloads for the four different scenarios, respectively.

Obviously, the distributed server architecture improves the runtime performance greatly compared to the original single-server Agentflow architecture within three of the four kinds of requests. Moreover, under different workloads, ranging from 50 to 2500 instances, the distributed server architecture can deliver a nearly constant response time, benefiting from its scalable architecture. This is a desirable feature for modern service-oriented systems which confront the incoming requests with the unpredictable and dynamical amounts of change, while being expected to maintain acceptable and stable response time. The get TaskOfCompany() request is a special case, which must get the task instances from all running PASE resources. Therefore, in some situations, it may take much longer time to finish than that in the original Agentflow architecture which involves only one centralized PASE resource.

Fig. 11 shows that 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 distributed server architecture is shorter than 4500 ms. This result indicates that the distributed server architecture proposed in this paper can effectively maintain an acceptable request response time under request loads of large variation.

The following presents the comparisons of the three different load metrics. As seen in Figs. 12–16, the arrival-rate mode and

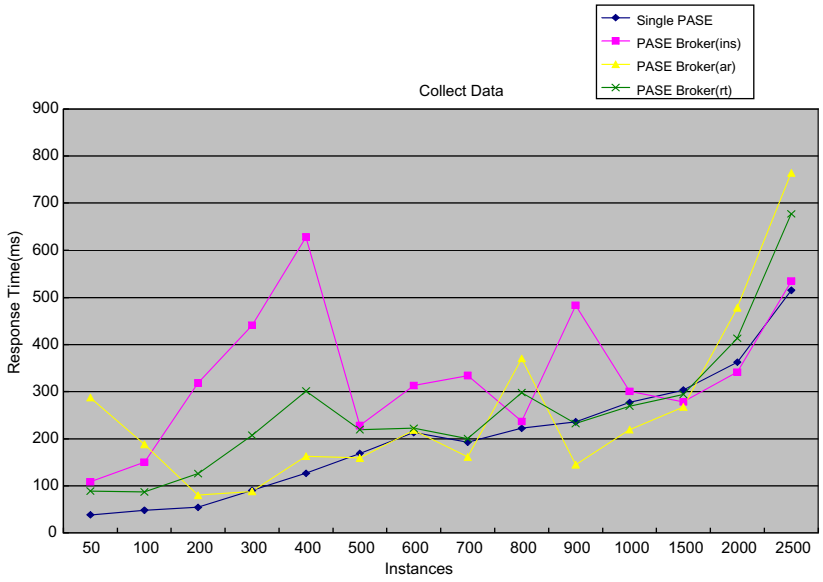

Fig. 10. Performance results of get TaskOfCompany().

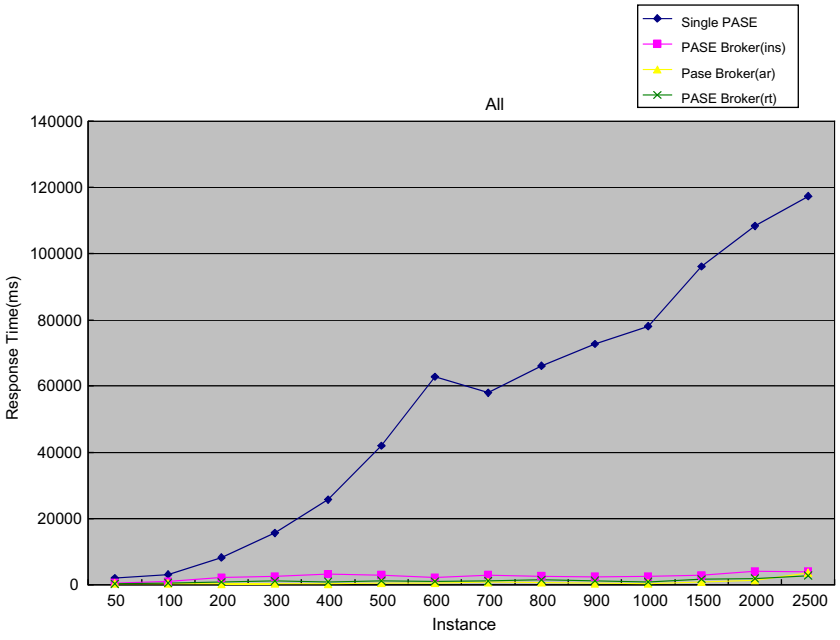

Fig. 11. Average response time of all requests.

the response time mode in general outperform the instance mode. The PASE broker with the arrival-rate mode performs best and delivers a shorter and more stable average response time than with the other two modes. However, the performance based on the arrival-rate mode or the response time mode could be influenced by the corresponding buffer sizes set in the performance monitor. Therefore, the performance based on these two modes need be studied further. For example, it might be useful for the comparison by investigating the effects of different buffer sizes. Another important issue, how to determine an appropriate buffer size, might be useful in delivering good and stable performance with the dynamic request dispatching algorithm.

The following presents the results of another experiment of larger scale. Under the highest workload, totally 10 PASE resources were included in the running pool to serve the incoming requests. Fig. 17 shows that the number of PASE resources included in the running pool changed with time to adapt to different workload levels. With the dynamic resource provisioning capability, Fig. 18 shows that the request response times under different workload conditions were kept as constant as possible. Fig. 19 illustrates the benefits of stable request response time brought by dynamic resource provisioning. The red line represents the request response time of the distributed server architecture and the blue curve is the request response time of the original Agentflow architecture. The request response time of the original Agentflow architecture increased very quickly as the workload grew and varied greatly under different workload conditions. Compared to that, the request response time of the distributed server architecture is nearly

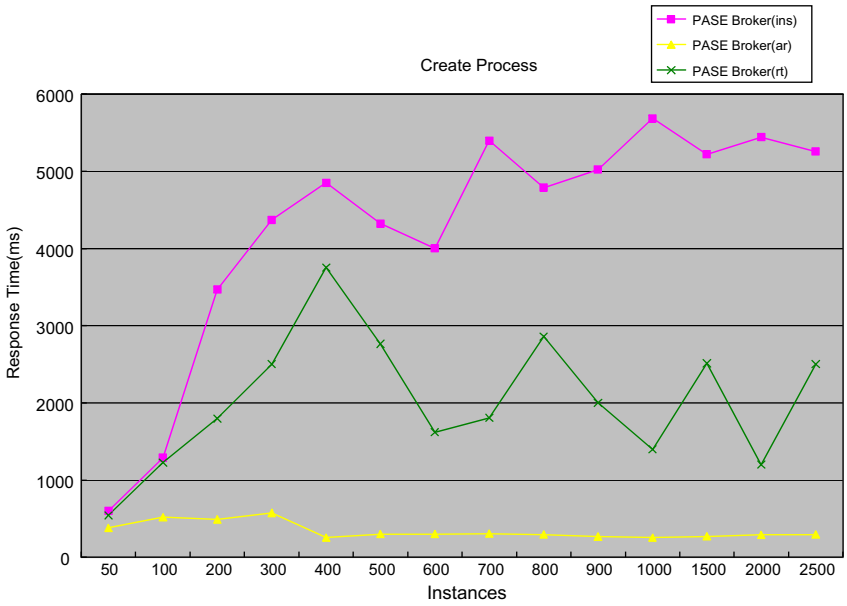

Fig. 12. Performance results of create Process().

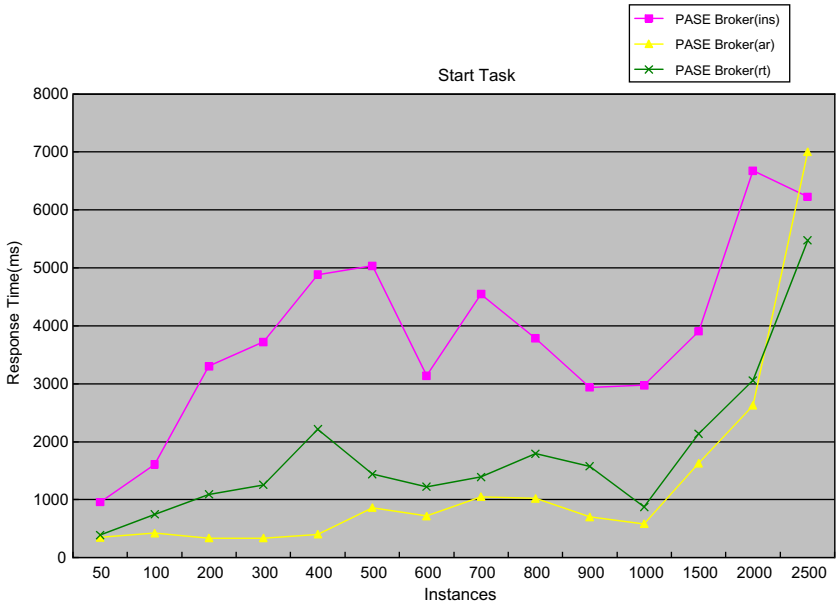

Fig. 13. Performance results of start Task().

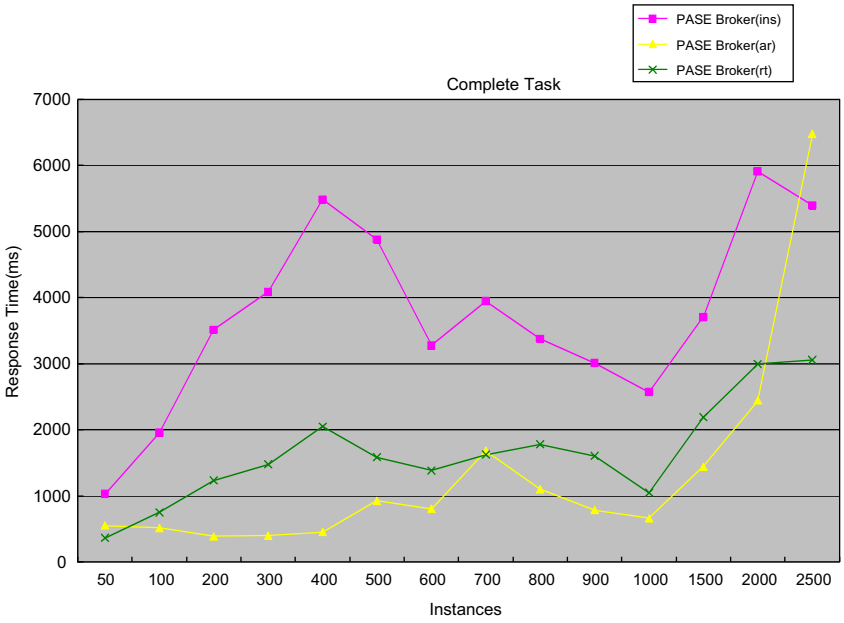

Fig. 14. Performance results of complete Task().

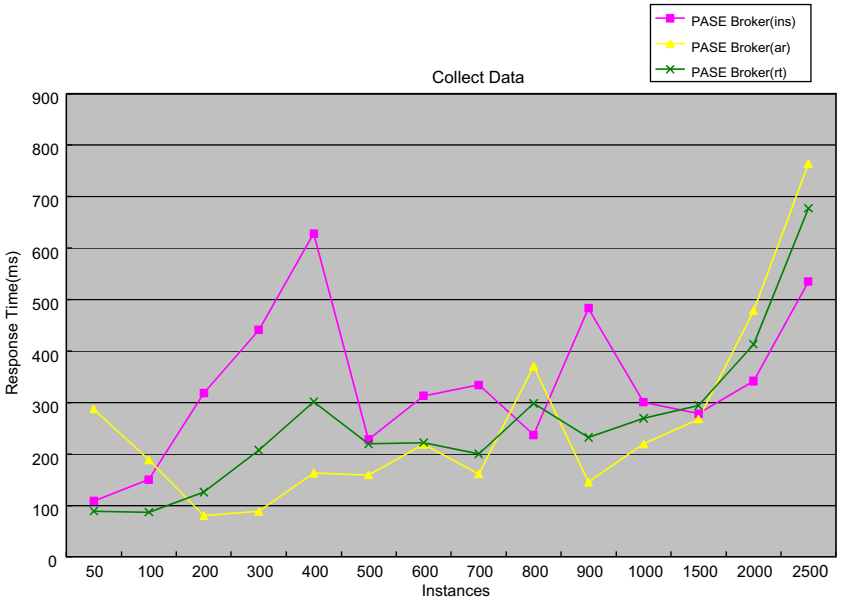

Fig. 15. Performance results of get TaskOfCompany().

constant throughout the whole experiment duration. In Fig. 19 we can also see that the red line ended much earlier than the blue curve. This means that the distributed server architecture finished processing all the requests earlier than the original Agentflow architecture. The above results demonstrate the potential benefits of the proposed distributed server architecture and dynamic resource provisioning mechanisms.

5. Conclusions and future work

This paper presents a distributed server architecture with dynamic resource provisioning mechanisms for building scalable BPM-oriented workflow management systems. We implemented a prototype system and made a series of experiments to evaluate the performance of the proposed architecture and mechanisms. The experimental results indicate that the proposed architecture is effective in handling the time-varying workloads in real world workflow management systems. The scalable architecture with the capability of dynamic resource provisioning can provide acceptable and stable request response time under a wide range of dynamic request workloads. This is a desirable feature for modern service-oriented systems which confront the incoming requests whose amounts are unpredictable and change dynamically, while being expected to maintain acceptable and stable response time.

Besides, some works might be worthwhile for improving the system performance further. For example, determining an appropriate

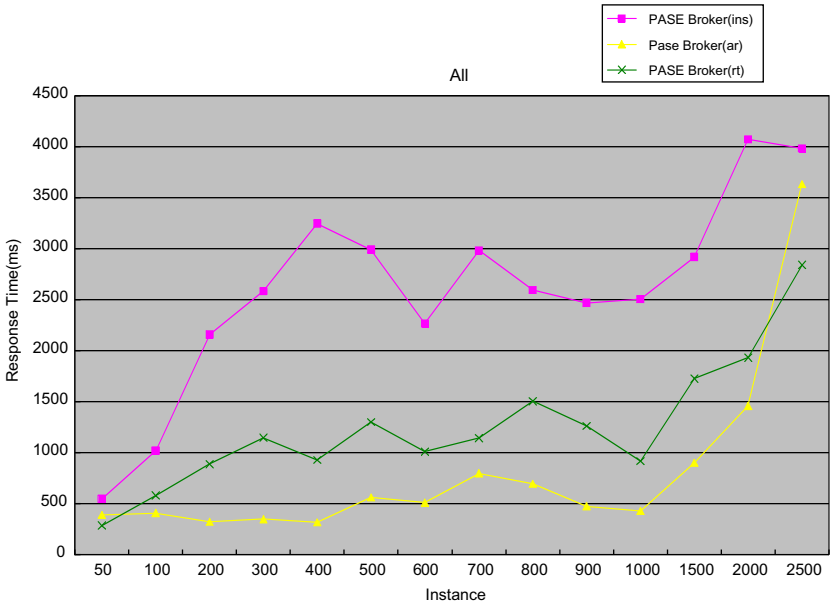

Fig. 16. Average response time of all requests.

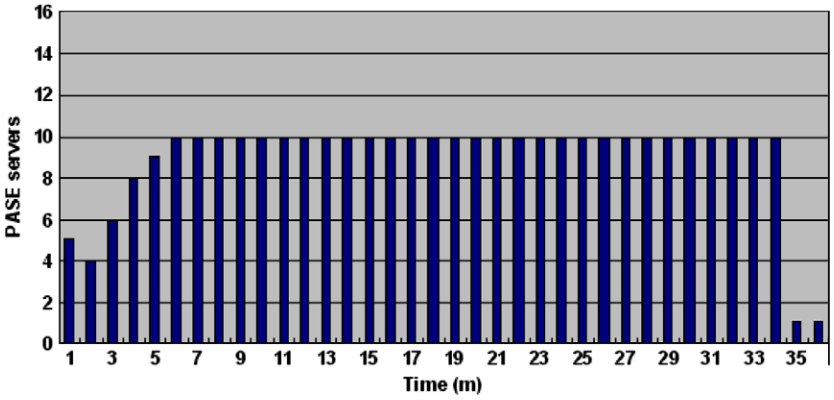

Fig. 17. Number of PASE resources used during experiment.

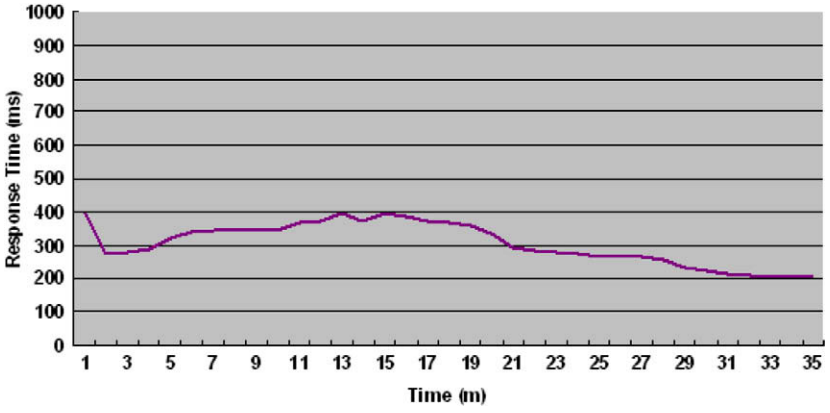

Fig. 18. Response time delivered by the distributed server architecture during experiment.

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

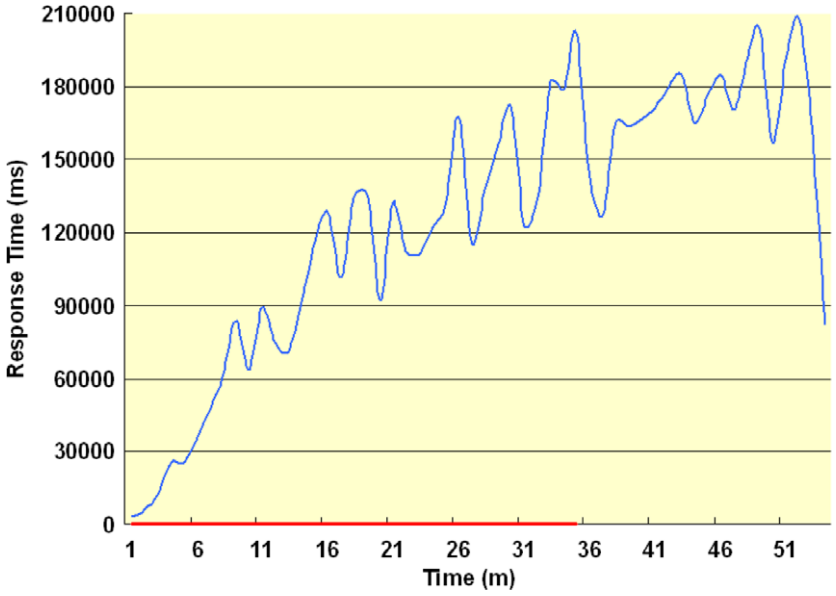

Fig. 19. Response times of the original Agentflow and the distributed server architecture.

requests to appropriate PASE resources for delivering good and stable runtime performance. Using history records to help predict future incoming requests are another promising approach to enabling the dispatcher for making more appropriate allocation decisions.

Acknowledgement

The authors would like to thank Hui Zhen Zhou for his support on conducting the experiments in this paper.

References

Agentflow system, 2008. Flowring Technology Corp., <http://www.flowring.com>.
Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludaescher, B., Mock, S., 2004. Kepler: Towards a Grid-enabled system for scientific workflows. In: Proceedings of Workflow in Grid systems Workshop in GGF10.
Baru, C., Moore, R., Rajasekar, A., Wan, M., 1998. The SDSC storage resource broker. In: Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research.
BPAF (Business Process Analytics Format), 2009. Workflow Management Coalition, <http://www.wfmc.org/bpaf.html>.
BPMN (Business Process Modeling Notation), 2009. Object Management Group, <http://www.omg.org/spec/BPMN/1.2/>.
Buyya, R., 1999. High Performance Cluster Computing: Architectures and Systems. Prentice Hall PTR. vol. 1.
Cao, J., Jarvis, S.A., Saini, S., Nudd, G.R., 2003. GridFlow: Workflow management for grid computing. In: Proceedings of Third International Symposium on Cluster Computing and the Grid.
Chiang, M.L., Lin, Y.C., Guo, L.F., 2008. Design and implementation of an efficient web cluster with content-based request distribution and file caching. Journal of Systems and Software 81 (11), 2044–2058.
Chou, S.J., Feng-Jian Wang, F.J., 2001. Constructing a Management System for a University Department, Master Thesis, National Chiao-Tung University.
Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C., 2001. Grid Information Services for Distributed Resource Sharing. In: Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing.
Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S., 2005. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Scientific Programming Journal 13 (3), 219–237.
Duan, H., Zeng, O., Wang, H., Sun, S.X., Xu, D., 2009. Classification and evaluation of timed running schemas for workflow based on process mining. Journal of Systems and Software 82 (3), 400–410.
Foster, I., 2002. The grid: a new infrastructure for 21st century science. Physics Today 55 (2), 42–47.
Foster, I., Kesselman, C., Tuerke, S., 2003. The Grid: Blueprint for a New computing Infrastructure, Morgan Kaufmann.
Gerasoulis, A., Yang, T., 1993. On the granularity and clustering of directed acyclic task graphs. IEEE Transactions on Parallel and Distributed Systems 4 (6), 686–701.
Globus Toolkit, 2008. The Globus Alliance, <http://www.globus.org>.
Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J., 2008. On the use of cloud computing for scientific workflows. In: Proceedings of Third International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES) in conjunction with Fourth IEEE International Conference on e-Science (e-Science 2008).
Hollingsworth, D., 1995. The workflow reference model, TC00-1003, The Workflow Management Coalition Specification.
Hsu, H., Wang, F., 2008. An incremental analysis for resource conflicts to workflow specifications. Journal of Systems and Software 81 (10), 1770–1783.
Krishnan, S., Bramley, R., Gannon, D., Govindaraju, M., Alameda, J., Alkire, R., Drews, T., Webb, E., 2001. The XCAT science portal. In: Proceedings of Supercomputing.
Laszewski, G.V., Amin, K., Hategan, M., Zaluzec, N.J., Hampton, S., Rossi, A., 2004. GridAnt: A client-controllable grid workflow system. In: Proceedings of 37th Hawaii International Conference on System Science.
Mathur, V., Apte, V., 2009. An overhead and resource contention aware analytical model for overloaded web servers. Journal of Systems and Software 82 (1), 39–55.
Mendonça, N.C., Silva, J.A.F., Anido, R.O., 2008. Client-side selection of replicated web services: an empirical assessment. Journal of Systems and Software 81 (8), 1346–1363.
MySQL HA/Scalability Guide, 2009. MySQL, <http://www.dev.mysql.com/doc/mysql-ha-scalability/en/ha-overview.html>.
OASIS (Organization for the Advancement of Structured Information Standards), 2009. <http://www.oasis-open.org/home/index.php>.
OMG (Object Management Group), 2009. <http://www.omg.org/>.
Prodan, R., 2007. Specification and runtime workflow support in the ASKALON grid environment. Scientific Programming 15 (4), 193–211.
Prodan, R., Fahringer, T., 2008. Overhead analysis of scientific workflows in grid environments. IEEE Transactions on Parallel and Distributed Systems 19 (3), 378–393.
Ranjan, S., Knightly, E., 2008. High-performance resource allocation and request redirection algorithms for web clusters. IEEE Transactions on Parallel and Distributed Systems 19 (9), 1186–1200.
Roure, D.D., Baker, M.A., Jennings, N.R., Shadbolt, N.R., 2003. The evolution of the grid. In: Grid Computing: Making the Global Infrastructure a Reality. John Wiley and Sons, pp. 65–100.
Shields, M., Taylor, I., 2004. Programming scientific and distributed workflow with Triana services. In: Proceedings of Workflow in Grid Systems Workshop in GGF 10.
Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., Wood, T., 2008. Agile dynamic provisioning of multi-tier internet applications. ACM Transactions on Autonomous and Adaptive Systems 3 (1), 1–39.
Wang, X., Du, Z., Chen, Y., Li, S., 2008. Virtualization-based autonomic resource management for multi-tier web applications in shared data center. Journal of Systems and Software 81 (9), 1591–1608.
Wei, J., Xu, C.Z., 2006. eQoS: provisioning of client-perceived end-to-end QoS guarantees in web servers. IEEE Transactions on Computers 55 (12), 1543–1556.
WfMC (Workflow Management Coalition), 2009. <http://www.wfmc.org/>.
WS-BPEL (Web Services Business Process Execution Language), 2007. OASIS, <http://www.docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.
XPDL (XML Process Definition Language), 2008. Workflow Management Coalition, <http://www.wfmc.org/xpdl.html>.

Yoo, M., 2009. Real-time task scheduling by multiobjective genetic algorithm. Journal of Systems and Software 82 (4), 619–628.
Zeng, Q., Wang, H., Xu, D., Duan, H., Han, Y., 2008. Conflict detection and resolution for workflows constrained by resources and non-determined durations. Journal of Systems and Software 81 (9), 1491–1504.

Ching-Hong Tsai received B.S. and M.S. degrees in Computer Science and Information Engineering from National Chiao-Tung University, Taiwan, ROC, in 1996 and 1998. His main research interests include software engineering process, software requirement analysis, distributed computing, and workflow technology. He is currently working on the Ph.D. degree in Computer Science at National Chiao-Tung University.

Kuo-Chan Huang received his B.S. and Ph.D. degrees in Computer Science and Information Engineering from National Chiao-Tung University, Taiwan, in 1993 and 1998, respectively. He is currently an Assistant Professor in Computer and Information Science Department at National Taichung University, Taiwan. He is a member of ACM and IEEE Computer Society. His research areas include parallel processing, cluster and grid computing, workflow computing.

Feng-Jian Wang graduated with M.S. and Ph.D. degrees from North-western University. Currently, he is a Professor in Computer Science Department, National Chiao-Tung University, HsinChu, Taiwan. He is an IEEE member. He has published more than 70 technical papers in international journals and conference proceedings. He is the founder of the Flowring Technology Corp. in Taiwan. His research areas include software engineering, network engineering, and programming language.

Chun-Hao Chen is currently a graduate student in pursuit of a M.S. degree in Computer Science at National Chiao-Tung University in Taiwan.