Dynamic Provisioning of a Parallel Workflow Management System

Ching-Hong Tsai
Department of Computer Science, National Chiao-Tung University, Taiwan
chtsai@cs.nctu.edu.tw

Kuo-Chan Huang
Department of Computer and Information Science, National Taichung University, Taiwan
kchuang@mail.ntcu.edu.tw

Feng-Jian Wang
Department of Computer Science, National Chiao-Tung University, Taiwan
fjwang@cs.nctu.edu.tw

Abstract

Most workflow management systems nowadays are based on centralized client/server architecture. Under this architecture, the response time of request might increase unacceptably when the number of users who login to the system increase quickly and a large amount of requests are sent to the centralized server within a short time period. Parallel server architecture could help to resolve the performance bottleneck of a single server. However, a static parallel architecture with a fixed number of servers is not efficient at resource utilization because the numbers of users and their requests usually change time by time, especially for big and fast changes. This paper presents an effective architecture of dynamic resource-provisioning and then the implementation for a parallel workflow management system. There are a series of experiments conducted and the results indicate that it is an effective approach to handling the time-varying workloads in real world WfMS.

1. Introduction

To fulfill the ever growing needs of business process automation, workflow management systems (WfMS) have been broadly adopted by many enterprises 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. Most workflow management systems work based on client-server architecture, where each system provides one single workflow engine and other tools such as database system to support the development and running of a workflow application. For example, *Agentflow*, a well-known JAVA-based WfMS developed by our laboratory and then Flowring co. [1] in Taiwan, works with this structure.

Obviously, the response time under such architecture is bounded by the computing power of a centralized server engine and the capacity of the database at least. 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. On the other hand, a fixed number of servers with a static parallel architecture might not be effective as expected because a big amount of changes of users, incoming requests, and the corresponding execution time might occur in a sudden in a real world WfMS.

This paper presents a grid architecture for a series of server engines to resolve the performance bottleneck of a WfMS, where each component WfMS runs with a single centralized server engine, such as Agentflow. The proposed grid architecture is focused on the utility computing aspect of grid computing [6,7,8,12], which is similar to the recently emerging concept of cloud computing. The computing resources in the grid are stable without frequent joining or leaving activities. Dynamic provisioning capabilities are the major concern for fulfilling the time-varying resource requirements of individual applications. We made an implementation of the proposed architecture and conducted a series of evaluation experiments. The experimental results indicate that our architecture is effective for handling the time-varying workloads in real world workflow management systems.

2. Background

The Agentflow system [1] is a JAVA-based WfMS with centralized client-server architecture. There are three main components in Agentflow:

978-0-7695-3471-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ISPA.2008.116

- 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.
- 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.

The database system inside Agentflow contains two repositories, *process definition repository* 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.

Some research projects deal with supporting computational workflows on grid systems. These systems manage the job dependencies and control the flows of jobs in a gird environment and are sometimes called grid workflow systems. These grid workflow system include GridAnt [11], Triana [13], XCAT [10], GridFlow [4], and Kepler [2]. Most of these grid workflow systems are focused on the support of scientific workflow applications. In this paper, we develop a dynamic resource provisioning architecture for supporting business workflow applications.

3. Scalable workflow computing with dynamic resource provisioning

This section extends Agentflow to a scalable workflow computing platform with dynamic resource provisioning. The scalable platform 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. PASE grid architecture

During the execution of a typical WfMS, Agentflow, there is a problem found. The request response time increases greatly when the requests arrive at one Agentflow at a high rate. It indicates that there is a bottleneck for Agentflow system. Based on the review of Agentflow system in above section, the single centralized server for the platform might be the source for the bottleneck. Therefore, we propose a scalable workflow computing platform, called PASE grid, which equips Agentflow with dynamic resource provisioning capability. The PASE grid architecture is shown in Figure 1 and its constituent components will be elaborated in the following.

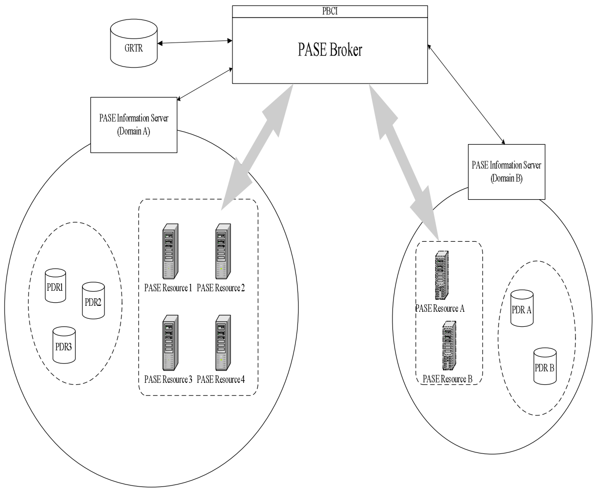

Figure1. PASE grid architecture

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.

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.

G*lobal 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 a PASE grid, managed by the PASE broker.

PASE information server (PIS) plays a role similar to the MCAT in Storage Resource Broker [3] or Grid Information Service (GIS) in Globus Tookit [9], 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 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 serve incoming requests. The *blocking* state indicates the failure of a PASE resource.

A PASE broker coordinates PIS's, PASE resources, PDR's, and GRTR. The architecture of a PASE broker is illustrated in Figure 2.

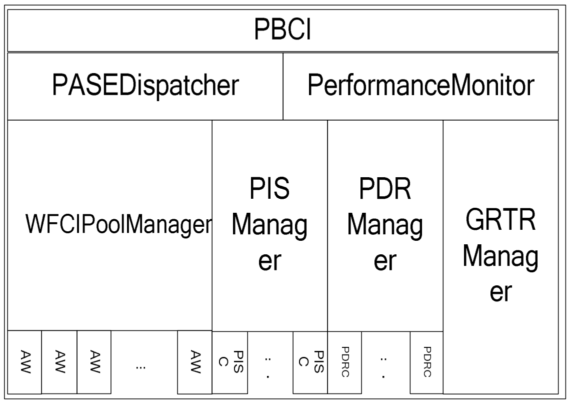

Figure. 2. PASE broker architecture

PISManager connects to and manages all PIS's. A PISC in Figure 2 is a connection from the PASE broker to a PIS. PISManager 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 PISManager sends replication request to all PIS's for replicating process definitions into their PASE resources. *PDRManager* connects to and manages all PDR's. A PDRC in Figure 2 is a connection from the PASE broker to a PDR. All the clients' requests of getting the process definition related data are handled by PDRManager. *GRTRManager* backups the completed workflow instances inside the PASE resource which is to be removed by the PASE broker.

WFCIPoolManager 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.

WFCIPoolManager 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.

PerformanceMonitor (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, PerformanceMonitor will inform WFCIPoolManager to find out more usable PASE resources from the PIS's in use under the order defined in PISManager, and create the connections to them. If there are no new PASE resources found, WFCIPoolManager 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 PASE grid.

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 1.

Table 1. 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

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.

3.2. On-demand resource provisioning strategies

This section discusses the resource provisioning strategies used in the PASE grid. Among the various kinds of user requests, *task manipulation* requests (TMRs) and *process enactment requests* (PERs) can benefit from this PASE grid architecture. On the other hand, the *data collection* requests (DCRs) would need a little bit longer time than those in the original centralized architecture.

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.

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.

PerformanceMonitor monitors the performance of each PASE resource in the PASE grid. It sends an event to WFCIPoolManager for adding new PASE resources or withdrawing some existing PASE resources when the entire PASE grid is overloaded or under-utilized. PerformanceMonitor 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, where both maximum and minimum are specified by the administrator. If the workloads of all PASE resources in the running pool exceed their maximum, the PASE grid is *overloaded*. On the other hand, if the workloads of all PASE resources in the running pool are lower than their minimum for a pre-defined time period, the PASE grid is deemed as *under-utilized*.

Once all running PASE resources are overloaded, WFCIPoolManager will try to discover computing resources outside and set them as available PASE resources for use. The resource addition is done gradually in order to reduce variation. WFCIPoolManager 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, WFCIPoolManager 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 can save the time for setting up a new PDR.

On the other hand, when the incoming requests decrease and the overall system has been under-utilized, the PASE grid will release a portion of the PASE resources for use by other demanding PASE brokers. WFCIPoolManager selects a running PASE resource and move it to the suspending pool. Corresponding to the above resource-addition mechanism, WFCIPoolManager follows a gradual-shrink policy, i.e., it withdraws the PASE resource with the least workload processing power in the running pool.

WFCIPoolManager periodically checks all the AW's in the suspending pool. For those AW's finished all workflow instances on them, it first informs the GRTRManager to backup instance data and then asks

PISManager to clear up the instance data as well as the process definition data in the PASE resources' databases. Finally, WFCIPoolManager disconnects these PASE resources from the PASE broker.

4. Performance evaluation

Based on the PASE grid architecture described in section 3, we have implemented a prototype system and conducted a series of experiments for performance evaluation. We set up a PASE grid consisting of four PASE resources. All PASE resources will use the same process definition repository in the experiments.

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 process definitions adopted in the experiments are real cases obtained from [5], 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 & network center, and 5) laboratories. The services provided by these subsystems are defined and run on Agentflow. In the following experiments, we created 1,500 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.

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 grid and configure its corresponding PDR.

In the following experiments, the amounts of workflow instances range from 50 to 2,500, the request 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 performances of four different scenarios, including a single PASE server in the original Agentflow architecture and the PASE grid architecture with three different load metrics, respectively.

Figure 3 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 PASE grid architecture is shorter than 4,500 ms. This result indicates that the PASE grid architecture proposed in this paper can effectively maintain an acceptable request response time under request loads of large variation.

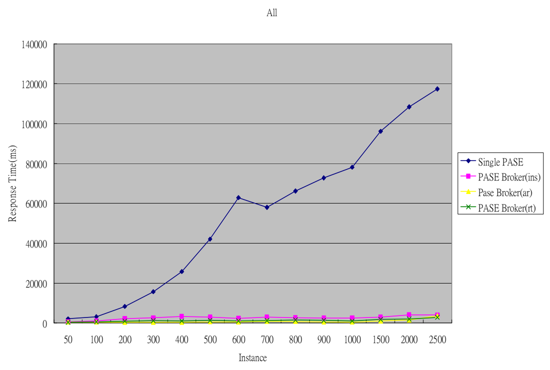

Figure. 3. Average response time of all requests

As seen in the above figure, the arrival-rate mode and 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 in the future work.

5. Conclusions and future work

This paper presents a dynamic architecture of resource provisioning and a set of algorithms to construct a (parallel) workflow management system. We implemented a system and made a series of experiments to evaluate this implementation. The results indicate that the proposed architecture is an effective approach to handling the time-varying workloads in real world workflow management systems. The scalable platform 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 with the amounts of unpredictable and dynamical change, 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 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. Using history records to help predict future incoming requests is another promising approach to enabling the dispatcher for making more appropriate allocation decisions.

6. References

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

[2] 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.

[3] 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*.

[4] Cao, J., Jarvis, S. A., Saini, S., Nudd, G. R., "GridFlow: Workflow management for Grid computing", *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid*, 2003.

[5] Chou, S. J., Feng-Jian Wang, F. J., *Constructing a Management System for a University Department*, Master Thesis, National Chiao-Tung University, 2001.

[6] 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.

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

[8] Foster, I., Kesselman, C., Tuerke, S., *The Grid: Blueprint for a New computing Infrastructure*, Morgan Kaufmann, 2003.

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

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

[11] Laszewski, G. V., Amin, K., Hategan, M., Zaluzec, N. J., Hampton, S., Rossi, A., "GridAnt: A client-controllable Grid workflow system", *Proceedings of 37th Hawaii International Conference on System Science*, 2004.

[12] Roure, D. D., Baker, M. A., Jennings, N. R., Shadbolt, N. R., "The Evolution of the Grid", *Grid Computing: Making The Global Infrastructure a Reality*, JohnWiley& Sons, 2003, pp. 65–100.

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