Load Balancing In Distributed Workflow Management System

Li-jie Jin
HP Laboratories
Palo Alto, California, U.S.A
1 (650) 236 8093
ljjin@hpl.hp.com

Fabio Casati
HP Laboratories
Palo Alto, California, U.S.A
1 (650) 236 8437
casati@hpl.hp.com

Mehmet Sayal
HP Laboratories
Palo Alto, California, U.S.A
1 (650) 857 4497
sayal@hpl.hp.com

Ming-Chien Shan
HP Laboratories
Palo Alto, California, U.S.A
1 (650) 857 7158
shan@hpl.hp.com

ABSTRACT

Workflow Management Systems (WFMS) play a very important role in constructing today's e-commerce environment through automating intra-enterprise business processes and inter-enterprise services. To handle the rapidly changing business environment and global competition, WFMSs should have flexibility and scalability to meet the business requirement and to quickly introduce new and efficient business services. Achieving load balancing is essential to ensure scalability in a distributed WFMS. In this paper we discuss load-balancing technology for distributed WFMSs. First, we introduce a workflow load index to measure load level of workflow engines. Then we present a WFMS cluster architecture with a load balancing subsystem. We compare the performance of round robin versus load-aware scheduling under the same load pattern. The experimental results show that the load index that we define in this paper is a good indicator of the load level in a distributed WMFS. The results also suggest that the load-aware scheduling algorithm can distribute workload fairly on heterogeneous WFMSs; instead, the round robin scheduling can only guarantee load balance in uniform WFMS with uniform workload and resource capabilities.

Keywords

Load balancing, workflow, load index, business process

1. INTRODUCTION

Workflow Management Systems (WFMSs) play a very important role in constructing today's e-commerce environment by automating intra-enterprise business processes and inter-enterprise services. A WFMS combines technologies of distributed computing, Internet, database and business process management. It provides the capabilities to define, develop, execute and monitor business processes [11][9]. To fit with the rapidly changing business environment and with the high-varying load of e-business applications, WFMSs should be scalable and should provide the required flexibility to cope with peaks in the system load. For example, many commercial webs-based applications are supported by workflow technology [10]. Such applications must execute hundreds of thousands business processes everyday. It is possible that many customers access those sites in a very narrow time interval, or that the total amount of business transactions increases dramatically, due to the introduction of new products or to marketing campaigns. The workload may exceed the design capacity of a single workflow engine. Simply duplicating installation of WFMS cannot guarantee that the performance of WFMS scales when the equipment and management investment increases.

To solve this problem, we need to distribute the workload across multiple workflow engines so that the WFMS can maintain acceptable performance level under heavy load conditions. Our objective is to balance the workload in such a WFMS is to ensure performance scalability. WFMS load balancing mechanism with suitable resource configuration can also introduce many interesting features, such as quality management of business services, service reservation, and prioritized execution of selected business processes.

Load balancing algorithms and implementation technologies are long time research topics for researchers in various fields like distributed and parallel computing [19] [13][16][17], database systems [14], middleware systems [15][1], web server cluster [4][2][5], network flow control, and process management. To balance the workload in a generic distributed system, we first need to describe the workload level with suitable indexes; second, we need to have current load information of available processing units; third, we should have a scheduling strategy for distributing the work among processing units in order to meet pre-defined performance criteria; finally, we need to appropriately interface the load balancing components with the processing units so that the system architecture is stable when load balancing strategies are changed.

Load balancing for WFMSs share many challenges with distributed computing systems, distributed database systems and web server clusters. However, WFMSs have their own features that should be considered in balancing its workload. First, the workload of a WFMS is represented by business processes and their activities. The WFMS should be

SAC 2001, Las Vegas, NV

concerned not only with the performance of entire business processes, but also with the performance of the individual activities and/or sub-processes that belong to those processes. Second, the execution of business processes may involve humans and external applications. Unlike human activities in common web applications, human workers interact with WFMS not only as clients of the system, but also as service providers. A load model for WFMS should be able to cover the load impacts of both human and automated service providers.

In this paper, we present a workflow load index and a distributed WFMS architecture. This distributed WFMS architecture includes a *Business Process Unit* cluster and a *load balancing* sub system. Based on the load index, we propose a load aware Process Unit scheduling algorithm. Then, we compare its performance with that of a round robin Process Unit scheduling algorithm. The paper is organized as follows. Section 2 describes the process model and the architecture of our testing environment. Section 3 proposes a load model for WFMSs. Section 4 discusses a distributed WFMS architecture with load balancing capability. Section 5 gives performance results of two load-balancing policies on our prototype WFMS cluster. Finally, section 6 presents our conclusions and future research directions.

2. A WFMS PROTOTYPE

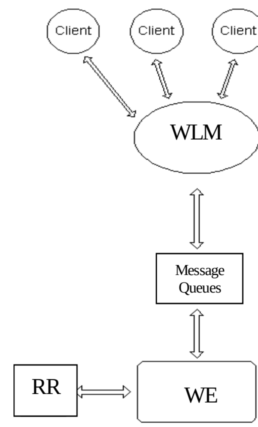
2.1 Process Model

The process model of a workflow system outlines essential process definition elements and their relationships [18] [3]. A workflow is a graph composed of *work nodes* that represent work activities to be executed and arcs that defines execution dependencies among activities. Each node is associated to a *service* to be executed, that includes a *resource rule* that defines who should perform the service as well as the definition of the input and output data. A workflow definition also includes nodes that denote the starting and ending point of the process, called *start node* and *end node* respectively. A process is considered completed when there are no more active nodes and no other node can be scheduled for execution.

2.2 WFMS Architecture

Fig. 1 displays a high level view of the testing WFMS architecture. The Workflow Engine (WE), the Worklist Manager (WLM), and the Role Resolver (RR) are the main components. The WE controls the execution of a process. It steps through definition of a process to determine the sequence in which activities are performed. The RR executes resource rules and assigns activities to the selected resource for execution. The WLM is a client management component, which acts as a work queue from which humans and applications can retrieve work items to be executed. The WLM joins WE to form a Process Unit (PU). The WLM picks up work items from worklist queues and enables their access to client applications. When a client notifies the WLM that he completed a work item, the Worklist Manager puts this work item into a *Send queue*, from which the WE will pull out completed work items.

The WLM guarantees that clients can only access those worklists that are assigned to them by WFMS administrator. Several WLMs can be included in a WFMS configuration. Enabling multiple WLMs configuration eases the administration of WFMS and enhances the mapping ability between organization model and business process model. Clients who belong to different organizations, have different authorizations, or have different type of service abilities can be assigned to different WLMs. Multiple WLM installation also improves system performance when a WFMS runs business processes that involve many interactions between a WE and activity executors. The RR is the role management component. It receives role description rules from the WE, read in the rule definition files. By running those rules, the RR resolves an address of the role that will carry out the activity and return this address to the Workflow Engine.



WLM: *Worklist Manager;*
RR: *Role Resolver;*
WE: *Workflow Engine.*

Fig. 1 Major components of the testing WFMS.

3. WFMS LOAD MODEL

Before considering how to balance workload in a distributed workflow management system, we need to define what exactly the workload of a WFMS is. We should know how to describe workload with load indexes, and how to collect the load information from Workflow Engine.

The major purpose of having a WFMS is to automate business process. Therefore, the basic workload of WFMSs is the business process. As we mentioned in section 2, business processes are composed by various business activities. The execution time of a business process is also composed by execution time of all its scheduled activities.

The execution time of an activity includes two major parts: the time for the resource to complete his job and the time for the WE to process the activity completion messages and schedule the next one. When a new activity is scheduled, the WE retrieves the definition of the activity, finds out the service associated with this activity, asks RR to execute the resource rule of the service and return the address of the resource, and finally sends the work item to the worklist of this resource. However, the *execution* latency of each activity in a WE may vary considerably. This is because that if the number of active process instances running in the WE changes, the lengths of process instance queue, activity queues and message queues will also change. These changes have direct impact on the engine response time for coming activity completion messages.

The candidate load parameters of WFMS include process execution time, active process instance number and WE latency between two adjacency activities of a process instance.

The execution time of a business process in a WFMS is an important performance parameter of both process clients and the WFMS. If multiple process instances are running simultaneously, the WFMS resources are shared by those instances. When the instance number increases, the shared execution time of each process instance in the workflow engine will decrease in a certain time period. Suppose that a process P is composed by a set of activities, WN_i, we have P = { WN_i | i = 0, 1, ..., n-1}. For a particular instance of this process, the execution path includes m activities. For each activity wn, we have wn $\in$ P. It is possible that m > n since some of the activities on the execution path may be executed more than once due to loops in the process. Let T_{proc} be the execution time of a business process instance PI of P, T_{wn} be the interval between start time of the previous activity in PI and start time of the current activity in PI. The last activity of each PI is responsible for returning process results and cleaning system. Its execution time can be ignored. Then we have $T_{proc} = \sum_{j=0}^{m-1} T_{wn_j}$. Let $T_{wn\_resource}$ be the resource role execution time of an activity, T_{wn_engine} be the engine execution time of the same activity, we have the execution time of the activity $T_{wn} = T_{wn\_resource} + T_{wn\_engine}$. The T_{wn_engine} is measured through subtracting the resource execution time of the first activity from the interval between two adjacency activities.

In our tests, when the number of active process instances increases, T_{proc} increases. T_{proc} reflects the workload level trend in some degree. It could be a WFMS load index if it is only used to measure load level of a WFMS with a single workflow engine or distributed WFMS with uniform WEs. When comparing the load level of WFMSs running on heterogeneous platforms, or having different configurations, this parameter may lead to wrong conclusions of relative workload. A slower system with less active process instances will show a lower load level than that of a much faster system with more active process instances. It is not always true that the system with lower load level indicated by this load index can offer higher service capacity than the system with higher load level indicated by the same load index. Another limitation of using T_{proc} as load index is that it cannot reflect any load information about business activities.

The WE execution latency between two adjacency activities of a process instance, T_{wn_engine}, describes the execution time of an activity on the WE. Our testing results indicate that change directions of this parameter reflect the same change trend of number of active process instances.

We observed various timing parameters of executing activities by running a sample business process in the testing WFMS. Fig. 2 shows trends of these parameters of different process instances (PIs). In the test, the process initiator started 400 process instances one after another in an arrival rate at 600 PIs/hour. All resources involved in this process are Java client programs. These Java clients check fields of work items and modify some fields with timing information for measuring purpose. The resource execution time for activities of each process instance during the test is set to be identical. For each process instance, the first process client initiates the process by invoking the initiator of the sample process. It records the process startup information in its work item. The second client responds to the first activity. It checks the request of the initiator and calculates timestamp differences. The third client responds to the second activity and does the same thing as the second client. Fig.2 illustrates the T_{wn} of the first two activities (T1i and T1f) in the test process of the first 50 process instances. While Txi (x = 1, 2, 3, 4) represent delays of execution the initiator, Txf (x = 1, 2, 3, 4) represent delays of execution the first activity. The execution time of a client, $T_{wn\_resource}$, is close to a constant during the test. Comparing to the value of T_{wn}, $T_{wn\_resource}$ can be ignored. Hence, from now on, we use T_{wn} to represent $T_{wn\_resource}$.

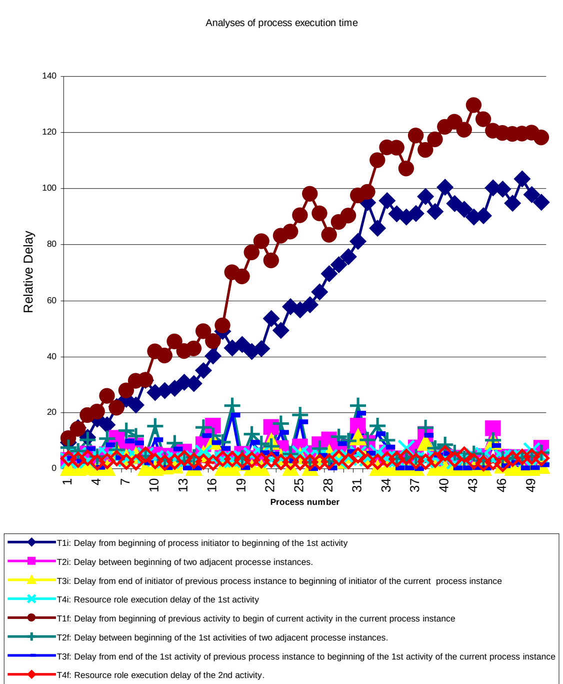


Fig. 2 Process execution delay (50 process instances)

Fig. 3 illustrates the relationship of T_{wn} and the number of active process instances in a 2000 process instance test with the same job arrival rate. It includes three parts. The top part shows that T_{wn} of the start node of the test process increases in a quasi-linear function of time when the job arrival rate was fixed. At time 100.5, all 2000 process instances had been started. There would be no new job being initiated from this point. The middle part shows the change of the number of active process instances in the same time interval. The maximum number of active process instances is 910 in a 2000 process instance test. The clock time when the WFMS reach this value is 100.5. This means that the T_{wn} can represent the increasing trend of load accurately. The bottom part of Fig. 3 shows the T_{wn} trend of the first activity. At clock time 100.5, T_{wn} began to decrease when there were no more process instances to start. At time 103, the WFMS completed the first activities of all 2000 process instances. This means that the T_{wn} can also represents the decreasing trend of load accurately.

According to the testing results, we have:

$$T_{wn} = N_{aPI} \times \lambda + C \tag{3.1}$$

where N_{aPI} is the number of active process instances in a PU, λ is the execution latency increase rate, and C is equal to the value of T_{wn} when there is only one active process instance in this PU. Both λ and C are performance parameters of a PU. C indicates the processing capacity of a platform on which the PU is running. λ is a function of the workload processing capacity of a certain PU configuration. T_{wn} indicates the current load level of a PU. It also points out how long a business activity of a process instance will be finished in the PU.

Based on this information, we introduce a load index for WFMS. Let L_{PE} be the load index of a PU. We define that: $L_{PE} = \sum_{i=0}^{k-1} T_{wn\_engine_i} \Big/ k$, while k is the number of completed activities of all active processes in the WE during a recent period of time. We define L_{PE1}, L_{PE3}, L_{PE5} as the average activity delays for the past one-minute, three-minute and five-minute respectively. When the system starts up, the initial value of L_{PE} is set to 0. When there is no activity being completed during the pre-setting time period, the value of L_{PE} keeps its last value.

There are four reasons for us to adopt L_{PE} as the load index of WFMS. First, according to our testing results, L_{PE} has a simple and close to linear function relationship with active process instances in the WE. Second, it is easy and cost effective to collect information from a WFMS to calculate the current value of L_{PE}. Third, this index can be used directly to estimate the execution time of a process instance with assumptions of execution branch prediction and expected resource role execution time for each activity. Fourth, L_{PE} is suitable to measure the load level of WFMSs running on platforms with different software and hardware configurations and capacities. With L_{PE} as the load index, we have an identical measurement to weight the load level of WFMSs. We can also predict the execution time of both active process instances and incoming process instances with the value of the current load index, no matter what kind of platform the WFMS is installed on and how many process instances are running in the WFMS.

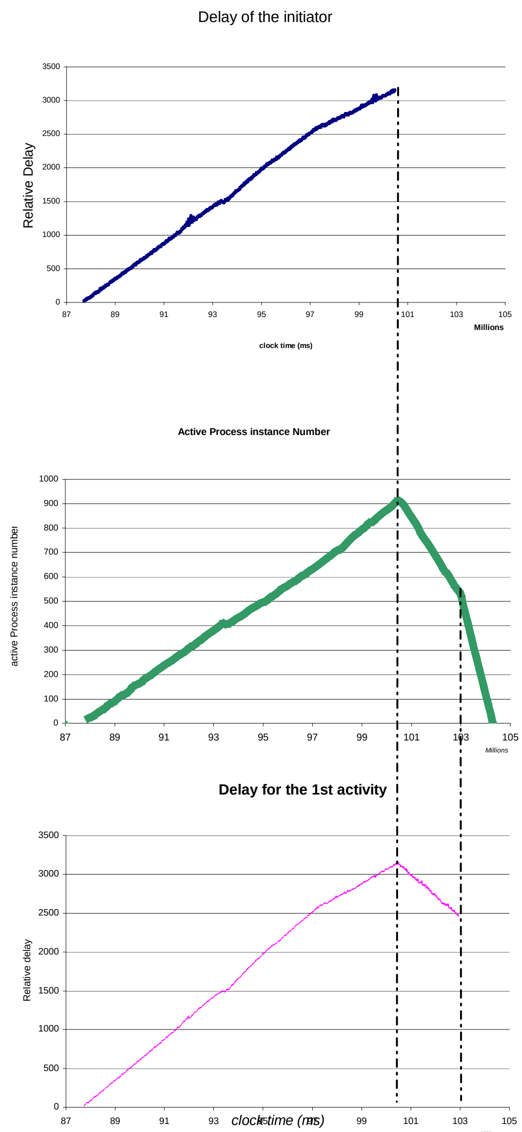


Fig. 3 Relationship between number of active process instance and Twn

4. LOAD BALANCING IN A WFMS WITH WF ENGINE CLUSTER

4.1 Objectives of load balancing in WFMS

The workload of a WFMS with a single Workflow Engine may exceed its capacity during the peak load periods. A

WFMS should be scalable in order to cope with the peak loads. The service capability of a WFMS should increase as the number of kernel components, such as Worklist Manager and Workflow Engines of the WFMS, increase.

The basic objective of balancing workload of WFMS is to make sure all system resources work effectively. A scalable WFMS can provide either a higher processing throughput under constant workload or a comparable processing throughput for heavier workload. The secondary objective is to improve the service quality of WFMS in a heavy workload environment. Our experimental results on the testing WFMS with a single Worklist Manager and Workflow Engine indicate that in a loaded system, if the job arrival rate is higher than the processing rate of the Workflow Engine, the latency between Workflow Engine's pick up of two consecutive activities in a process instance increases. Since the execution time of all involved resource roles of the process instance in our test is close to constant, the increased latency between consecutive activities is introduced by T_{wn_engine} only. These latencies extend the execution time of both the individual activities of active process instances and the active process instances themselves. WFMS can control the value of its L_{PE} through a load scheduling and balancing mechanism, which means that the engine latencies between consecutive activities of active process instances are under control. Therefore, the value of T_{proc} for each active process instance is predictable given that the average execution time of each involved WFMS resource role is predictable based on its qualification and historical performance data. When a pre-defined threshold of L_{PE} is reached due to high rate of incoming process instances, a WFMS can hold them in a queue that is outside of processing components. Eventually, the response time of those queued process instances will increase. However, that is the cost WFMS has to pay under a heavy loaded environment so that it can maintain a reasonable performance for all active process instances. The process instances that are blocked in the outside queue will generally have a change to start in the WFMS with a predictable response time from the time they are issued into the WFMS processing units. Setting up different thresholds of L_{PE} and adopting different process instance and Workflow Engine scheduling strategies allows WFMS to offer many new features to business process definitions, such as quality of service management for business process, process with deadline and priority requirement, and WFMS processing power reservation for some important business processes.

4.2 Design of Load Balancing Subsystem

We developed a load balance mechanism for WFMS based on the load index that is introduced in section 3. The load balancing mechanism is designed as a subsystem of WFMS. The Load Balancing Subsystem (LBS) has four main components: the Workload Monitor (LBWM), the Load Balancing Decision Maker (LBDM), the Job Scheduler (LBJS), and the distributed worklist management component (DWLM). We use the name Process Unit (PU) in order to describe a Workflow configuration that contains a Workflow Engine (WE), a Role Resolver, and a Worklist Manager(s) (WLM). A PU is a basic processing unit from the viewpoint of the LBS. The Load Balancing Subsystem supports multiple PUs in a WFMS configuration. A process instance is the basic unit of job to be scheduled among available PUs. The LBWM is responsible for collecting load information from each PU in a WFMS and distributing them among all PUs. The LBDM is the load balancing policy carrier and executor. The LBJS allocates process instance to a PU that is selected by the LBDM according to current load level information and load balancing strategies. Fig. 4 demonstrates the logic structure of LBS and the relationship between LBS and other WFMS components in a three-PU WFMS configuration.

LBWMs collect load parameters once every minute and calculate the current value of vector ($L_{PE1}, L_{PE3}, L_{PE5}$). Based on the load index definition, the essential parameters for calculating load level of a PU are the difference between start-times of consecutive activities and the resource execution times of those activities. There are various ways for the LBWMs to measure those load parameters. The LBWM can access WFMS log database and figure out the current value of load parameters. WFMS can embed into some PU-special fields of each work item the resource execution time and the timestamp of when this activity was started. Then the LBWM can fetch them from work items. The LBS can also run a special performance test process. The LBWM can also get the current load level information of this PU while measuring the performance of this process

The LBDM is the consumer of the load vector of each PU. The current load level of the PU satisfies the load policy that the LBDM current runs. The WFMS administrator sets up the load policy each time when the system is started. The policy can also be switched dynamically. The new policy will be effective in the next minute when the LBDM reads the updated load vectors from each PU. We will discuss more about load policies in section 4.4.

The LBJS has two functionalities. The first one is to redirect the process instance to a PU that is selected by the LBDM. The second one is to maintain a ready job queue (RJQ). In a heavy load environment, the LBDM may not always return a PU that can receive a new job without decreasing the performance of some other active process instances running on it to a value lower than a pre-defined threshold. Then all new incoming process instances will be held in RJQ. A WFMS administrator configures the size of a RJQ according to the peak load expectation of the WFMS and available system resources.

The Distributed Worklist Management (DWLM) is implemented on top of the WFMS Worklist Manager (WLM). Worklists are interfaces between business processes and their participators. The WLM is the WFMS component that implements the worklist and work item services between Workflow Engine and clients. The WLM connects to Workflow Engine through a group of queues. It has two major client classes: custom client applications that talk to the WLM with WFMS API; human clients that access their worklists from the WLM through web browser.

The need for to maintaining, manipulating and accessing distributed worklists comes from the need for supporting WFMS configured with multiple Process Units and load balancing among multiple PUs. In a multiple PU WFMS configuration, process instances of a business process may be scheduled to run on different PUs due to the system workload distribution and load balancing strategies. A participator of this business process may need to execute work items of the process instances generated from different PUs. It's not a practical to force a client to go through worklists of all PUs in the system to access his/her work items. The objective of the DWLM is to create a virtual, consistent worklist image for both process initiator and process participators, i.e. WFMS resource roles.

Business processes are deployed on all PUs. Every process belongs to a process group. All PUs have the same business process group structure in a multiple PU WFMS. Worklist definitions on all WLMs are identical, so are user accounts on all PUs. Users can access worklists from all WLMs theoretically, but each worklist has a default host WLM. From the users viewpoint, they access their worklists from the host WLM of their worklists only. Each PU has a DWLM component. All DWLMs are connected with each other through a message bus. From the viewpoint of the Workflow Engine, the DWLM acts just like a WLM. From the viewpoint of the Worklist Manager, the DWLM acts on multiple PUs just like a WLM acting on a single PU. The WLM accesses worklists from the DWLM instead of queues of each PU. The DWLM monitors the worklist and work item information from queues of each PU. The DWLMs have the knowledge of which worklist is hosted by which WLM. They exchange work items with each other periodically. The DWLM only creates a virtual consistent worklist image of a worklist on its host WLM since the overhead of maintaining the same single virtual worklist image of each worklist on every WLM is high.

4.3 Distributed Worklist Management

The consistency of distributed virtual worklists of a WLM is maintained by the DWLM in the same PU. When the DWLM gets a work item from a worklist of the WLR in the same PU and finds out that the host WLM of this worklist is in the same PU, it adds the work item to the end of the corresponding virtual worklist. If the host WLM of this worklist is not the local WLM, then the DWLM will send the work item to the PU that has the host WLM of this worklist. When the DWLM gets a work item from other DWLMs, it attaches a PU identifier (PUid) to the work item before adding it at the end of the corresponding virtual worklist. When WFMS client completes a work item, the default WLM will send the work item to a virtual Send Queue. This Send Queue is accessed by the local DWLM. The DWLM checks the attached PUid of the work item before dispatching it. If the PUid is null, then this work item is sent to the Send queue of the local PU. Otherwise, the work item is sent to the PU indicated by the PUid.

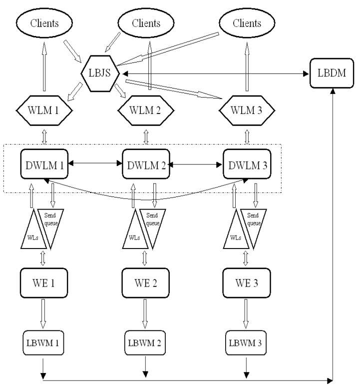


Fig. 4 Load Balancing Subsystem For WFMS

4.4 Load Balancing in Processing Unit cluster of WFMS

The objective of balancing workload in a distributed WFMS with multiple Process Units is to improve the performance of entire process instances and individual activities in process instances. In order to achieve these objectives, the load balancing sub system needs to allocate jobs to all PUs in the distributed WFMS in a way that PUs can have uniform workload according to the load index defined in section 3.

We assume the following: first, the basic scheduling unit of LBS is a process instance; second, the workload is scheduled during the initiation stage; and third, there is no active process instance migration between Process Units.

According to formula (3.1), in order to reduce T_{wn} of a Process Unit, we should either reduce the value of λ or control the number of active process instances in the Process Unit, N_{aPI}. The λ is a platform dependent parameter that the load balancing subsystem cannot control. In order to reduce λ, the system should have better hardware and network connection. We focus our efforts on controlling the number of active process instances in a Process Unit.

We present two Process Unit scheduling algorithms for workload control: Round Robin and Load-Aware. The Round-Robin algorithm schedules Process Units one at a time. It is effective when the processing power of each Process Unit is identical, the process types are the same, the initial load level of all Process Units are similar, and the resource role processing capacities are uniform. When any of these conditions is not satisfied, the workload of Process Units will not be distributed evenly.

The Load Aware PU Scheduling Algorithm (LAPSA) is our solution to above problems. It has the ability to balance the workload of Process Units according to their current load index without considering relative processing capabilities of Process Units.

Fig. 5 shows the working procedure of LAPSA. The components participating in the PU scheduling procedure include LBJS, LBDM, LBWM and all PUs in the WFMS.

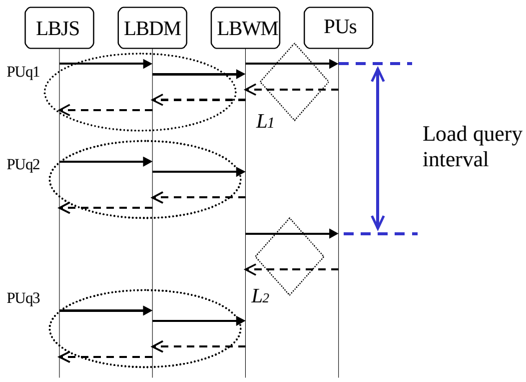


Fig. 5 Procedure of PU scheduling algorithm

The LBWM queries the load index parameters from each PU once in a fixed load query interval. The LBDM receives Process Unit scheduling requests from the LBJS. It then gets the most current load information from the LBWM. The LBDM returns a PU handler to the LBJS according to the current load index information and the load policy. So far, the LBDM always returns the PU with the lowest load index. The LAPSA reduces the new job arrival ratios of PUs with higher load index values, through sending new job to the PU with the lowest load index.

5. EXPERIMENTAL RESULTS

We implemented a prototype WFMS with a heterogeneous PU cluster and a load balancing subsystem based on a workflow management system that has only one PU. This prototype includes two PUs. One PU is installed on a slow Unix workstation. The other one is installed on a fast PC running NT server. A PC with Pentium III 600MHz CPU is used to generate workload, measure the load index and execute the tasks that are defined in work items on behalf of all resources that participate in the test business process.

We measured the activity response latency of 200 business processes on two different system configurations: WFMS cluster with Round-Robin PU scheduling policy and WFMS cluster with Load Aware PU scheduling policy.

Fig. 6 illustrates the trend of T_{wn} under Round-Robin scheduling policy. Each PU executes 100 process instances. Fig. 7 shows the trend of T_{wn} with the Load Aware scheduling policy. When the job arrival rate is 250 PI/hour, through Round-Robin scheduling policy, the NT server maintains an average T_{wn} around 12 relative delay unit. However, the slower WFMS installation on the Unix workstation still shows a linear increase of T_{wn} for process instances with even PI identifiers. When the Unix workstation receives the 100th process instance, there will be no new jobs to be started by the LBJS. Thus, the T_{wn} stops increasing for the Unix workstation. When the number of active process instances decreases, the T_{wn} decreases, too. The Round –Robin Algorithm distributes jobs to multiple PUs. However, this policy will cause problems when either the processing capabilities of PUs are not identical, or the load levels of PUs are not even. Approximately 50% of Process Instances that were processed by the Unix workstation have a T_{wn} value higher than 700 relative delay units with Round-Robin scheduling policy.

The Load Aware Scheduling policy sends job to the PU that has the lowest load level. This load level is measured by the load index that is defined in section 3. In our test, The LAPSA sends 132 out of 200 Process Instances to NT server and 68 Process Instances to the Unix workstation. The average T_{wn} of the NT server is maintained at approximately 18 relative delay units. The average T_{wn} of the Unix workstation is around 50 relative delay units. The balancing value of T_{wn} between these two PUs is around 28 relative delay units. Since the LBWM updates the load index for each PU in a fixed time interval, the LBDM may allocate jobs to a PU that just received large number of jobs before the LBWM refreshing its load index in the LBDM. That's why the Unix workstation has its average load index around 50 relative delay units. However, if the LBWMs update load index of their PU frequently, the load balancing subsystem will consume many CPU cycles and then will decrease the performance of all components of the WFMS. The faster PU processed more jobs with the LAPSA. There were few process instances that had their T_{wn} higher than 100 relative delay units. There is a significant performance improvement to the execution of 200 test process instances.

The reason for having fluctuation on curves in Fig 6 and 7 is that resource roles always get more than one work item from the WLM every time the WFMS is loaded. The process time of an activity whose work item was pre-fetched is shorter than that of the activity whose work item was received from the WLM right before it was processed.

6. Conclusions

In this paper, we proposed a load index for job scheduling in distributed Workflow Management System. We studied a distributed WFMS architecture with distributed worklist management mechanism and load balancing sub system. We then presented Load Aware Process Scheduling Algorithm (LAPSA) for WFMS. The performance improvement of the LAPSA was verified on a distributed WFMS. The test results show that our load index represents the load level of PUs accurately and it can handle the heterogeneities of WFMS.

The load aware balancing mechanism is suited for managing load of distributed workflow management system in order to achieve scalability. This mechanism gives WFMS the ability to support deadline and execution priority properties for business processes with multiple PUs. It also gives WFMS engine the ability to maintain quality of service for running process instances.

There has been many research efforts on defining load index and developing load-balancing policy for distributed systems [16][7][6][12]. Our work shares some common objectives with those efforts. However, our work focuses on load control technology within WFMS in which business processes are the workload of the distributed system.

The load balancing in distributed workflow management systems is an open research area. Our future research directions will include: ensuring scalability in large scale distributed WFMS with PU cluster through improving load measurement and job scheduling technologies; enabling new features for business process definition through load control and re-configuring the system dynamically.

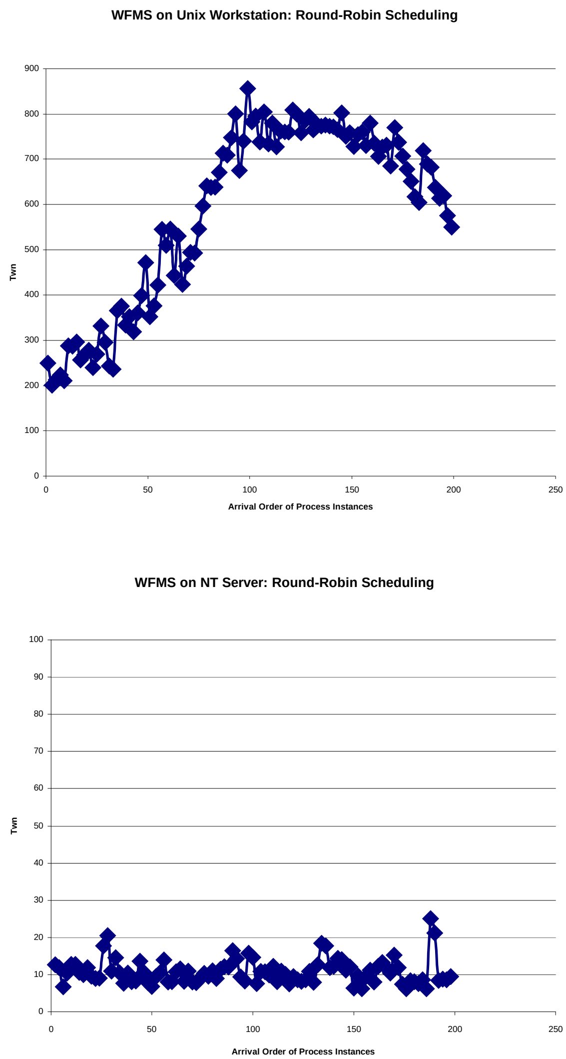


Fig. 6 Twn/job order relation under Round-Robin Scheduling.

(200 Process Instances, Job arrival rate = 250 PI/hour, 100 on NT Server, 100 on the Unix workstation)

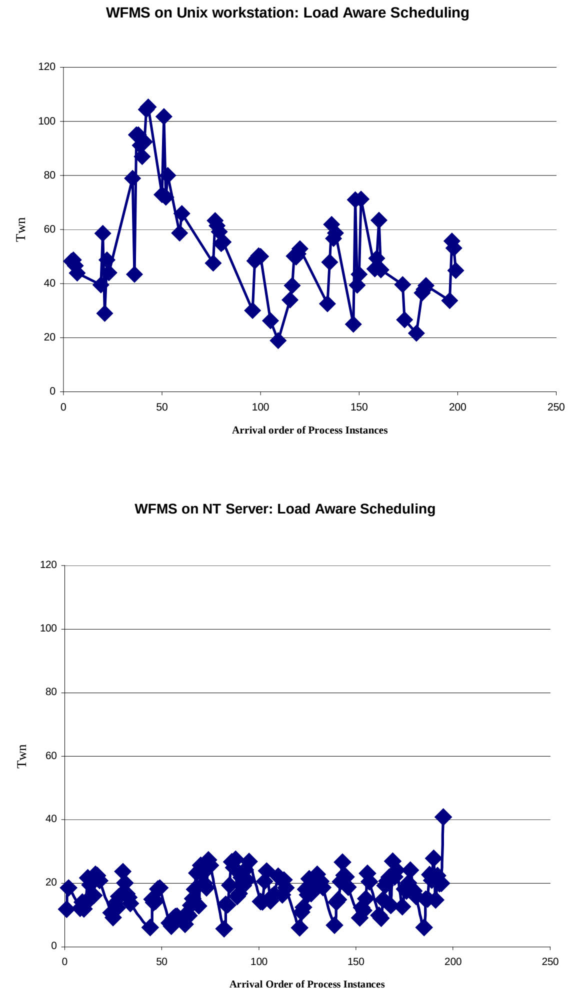


Fig. 7 Twn/job order relation under Load Aware Scheduling.

7. REFERENCES

[1] Andreas, B. Simplify with COM+ load balancing, Visual Basic Programmer’s Journal, Vol. 9, No. 11, pp30-34. 1999.

[2] S. Baker and B. Moon, Distributed Cooperative Web Servers, Proceedings of the 8th International World Wide Web Conference, 1999

[3] Fabio Casati, Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions , Ph.D. thesis, Politecnico Di Milano, 1998.

[4] O. Damani and P. Chung and Y. Huang and C. Kintala and Y. Wang, Techniques for Hosting a Service on a Cluster of Machines, Computer Networks and ISDN Systems, vol. 29, 1997.

[5] R. Farrell, Distributing the Web load, Network World, 1997

[6] D. Ferrari, S. Zhou, A load index for dynamic load balancing, 1986 Proceedings of Fall Joint Computer Conference, Nov. 2-6, 1986, Dallas, Texas.

[7] Susan F. Hummel, J. Schmidt, R.N.Uma, J. Wein, Load-Sharing in Heterogeneous System via Weighted Factoring, Eighth Annual ACM Symposium on PARALLEL ALGORITHMS AND ARCHITECTURES, SPAA'96, June 24-26 1996, Padua, Italy.

[8] E. Katz and M. Butler and R. McGrath, A scalable HTTP server: The NCSA prototype, Computer Networks and ISDN Systems, Vol. 27, 1994.

[9] Peter Lawrence, Workflow Handbook 1997, Workflow Management Coalition , Jan. 1997.

[10] http://Letsbuyit.com

[11] Frank Leymann, Dieter Roller, Production Workflow: concepts and Techniques. Prentice-Hall, Inc. 2000.

[12] P. Mehra, B.W. Wah, Automated learning of load-balancing strategies in programmed distributed systems, International Journal of Systems Science, Vol. 28, no. 11. pp 1077-1099.

[13] Dejan S. Milojicic, Milan Pjevac, LoadBalacing Survey, Proceedings of the Autumn 1991 EurOpen Conference, Sept. 1991, Budapest, Hungary.

[14] Erhard Rahm, Dynamic Load balancing in Parallel Database Systems, Proceeding of Euro-Par 96 Conference, LNCS, Springer-Verlag, Lyon, Aug. 1996.

[15] Thomas Schnekenburger, Load Balancing in CORBA: A Survey of Concepts, Patterns, and Techniques, The Journal of Supercomputing, Vol. 15, Issue 2, 2000.

[16] Niranjan G. Shivaratri, Mukesh Singhal, A load index and a transfer policy for global scheduling tasks with deadlines, Concurrency: Practice and Experience, Vol. 7(7), 671-688, Oct. 1995.

[17] Jeffery WestBrook, Load Balancing for Response Time. Journal of Algorithm 35, 1-16 (2000).

[18] WfMC, Interface 1: Process Definition Interchange – Process Model, Ver. 1.1, Oct. 1999.

[19] S. Zhou and D. Ferrari, A Trace-driven Simulation Study of Dynamic Load Balancing, Trans. On Software Engineering, 14(9), pp 1327-1341, IEEE (Sept. 1988).