論文 / 著書情報
Article / Book Information

Title	Activity Scheduling in Web-Service Based Workflow Managementfor Balancing Load and Handling Failures
Author	Hideyuki Katoh, Takashi Kobayashi, Haruo Yokota
Journal/Book name	Proc of 2006 International Workshop on Future Mobile and Ubiquitous Information Technologies (FMUIT'06), Vol. , No. , pp. 245-248
Issue date	2006, 5
DOI	10.1109/MDM.2006.28
URL	http://www.ieee.org/index.html
Copyright	(c)2006 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

Activity Scheduling in Web-Service Based Workflow Management for Balancing Load and Handling Failures

Hideyuki Katoh† Takashi Kobayashi‡ Haruo Yokota‡,†
† Graduate School of Information Science and Engineering, Tokyo Institute of Technology
‡ Global Scientific Information and Computing Center, Tokyo Institute of Technology
katoh@de.cs.titech.ac.jp, tkobaya@gsic.titech.ac.jp, yokota@cs.titech.ac.jp

Abstract

In a workflow management system, appropriate allocation of its activities greatly contributes to the improvement of its efficiency. We have proposed OXTHAS, a load-balancing method of scheduling the activities in workflow management systems using Web-services. The OXTHAS makes the activities to be assigned to appropriate executors based on the estimation of their processing capacity using execution histories in workflow engines. In this paper, we propose a failure-aware re-scheduling method for the OXTHAS using process time-outs under network and system failures. To allocate activities appropriately, the estimated processing capacity and time-out duration are re-calculated with consideration of the penalty for a failure when a process time-out occurs. We then evaluate the effectiveness of the proposed methods through simulations.

1. Introduction

The Web-services architecture is now attracting interests extensively. It allows the connection of widely distributed and heterogeneous systems chiefly with the HTTP and SOAP protocols. The strength of the Web-service technology mainly resides in its ability of surpassing incompatibilities in computing environments. Moreover, there is a noticeable tendency towards the Web-service based workflow management supported by researches on specifications such as BPEL4WS [1]. However, the load-balancing or failure handling of the Web-service based workflow management is slightly mentioned but not in detail in these specifications.

To cope with this issue, we have proposed the *OXTHAS (Observed Execution Time History and Activity Size based scheduling)*, a novel scheduling strategy for load-balancing in Web-services based workflow management [4, 5]. In the OXTHAS, the estimated processing capacity is calculated on the base of history of the previous execution of the different activity instances in terms of processing load size and execution time, as observed by each workflow engine. An activity instance is allocated to an appropriate executor by a workflow engine based on the estimated processing capacity and the processing load index. But, we have not made mention of the failure handling in it yet.

In this paper, we propose a re-scheduling method for handling network and system failures in the OXTHAS using process time-outs. In the proposal method, the estimated processing capacity and time-out duration are re-calculated with consideration of the penalty for a failure, to allocate activity instances appropriately under the Web-services and network failures.

2. Supposed Workflow Management Model

2.1. Workflow Management Architecture

The workflow management system (WFMS) architecture we assume in this paper mainly consists of workflow engines and executors. The workflow engine controls and manages the process and activity instances. A process represents a series of works and is composed of some activities, meaning a part of the works [2]. The outline of its internal functions is described as follows:

Workflow engine maintains a queue in which the clients insert the process instances and a queue in which the executors insert the activity instances that are processed. The workflow engine has the information of the process instances.
The workflow engine pulls out an activity instance from the queue and inserts the activity instance in the queue of executors to process it. Moreover, it inserts the activity instance in the queue of executors to inform the executors of ACK/NACK. If multiple requests of the same activity instance occur, the last request is adopted.

Executor maintains a queue in which the workflow engines insert the activity instances. The executor pulls out an activity instance from the queue and processes

the activity instance. When it finishes processing of the activity instance, it inserts the activity instance in the queue in the workflow engine.

It becomes possible to perform load-balancing of the whole system by defining a strategy by which the executors are being assigned the different activities instances. Here, we assume that two or more workflow engines exist and each workflow engine is premised on the ability of arbitrary sharing the executors. Moreover, we assume that each workflow engine knows to which executor the activity instances under management are passed.

2.2. Web-service based Workflow Management

In what follows, we consider a Web-services based WFMS and we build on the following two characteristics:

(1) The processing of an activity instance by an executor is a Web-service.

(2) Each workflow engine cannot know the inside information of executors.

First, it is necessary to consider the processing capacity for each activity, since an executor may provides more than one kind of the processing of activity instances as a Web-service. It is furthermore necessary to assume that the processing time is affected by the fact that the input file size of each activity instances is not constant. In this paper, it is proportional to the file size. In addition, we should consider the environment of provider, network and so on.

Next, if we take into consideration the possibility that multiple WFMS may coexist, then a number of workflow engines may share the same executor(s) through the Internet. Hence, a workflow engine cannot know the queue length of executors but can know only the time from the leaving of activity instances to the returning of them since requests of activity instances in executors' queue belong to multiple workflow engines. Note that the time includes processing time, queueing time, network delay, and so on.

3. OXTHAS Re-scheduling

3.1. OXTHAS Scheduling Strategy

In this section, we briefly explain the OXTHAS scheduling strategy we have proposed. Please refer to [4, 5] for the details of the OXTHAS scheduling strategy.

The OXTHAS-N dispatches each activity instances to top N executors capable of processing the activity. To estimate the performance of each executor for a certain activity, we introduce a measure, *Estimated Processing Capacity (EPC)*, which is calculated as the mean value of workload for each unit time, using execution history in the workflow engine. We defined the EPC, $c_{k,j}$, for the activity instance a_j assigned to the executor e_k, as follows:

$$c_{k,j} = \frac{\sum_{i=1}^{l} (\frac{s_{i,j}}{t_{i,j}} \cdot m_{i,j,k})}{\sum_{i=1}^{l} m_{i,j,k}}$$

where i is the process instance number for $P = \{p_1 \dots p_l\}$, $t_{i,j}$ is the observed execution time of the jth activity instance a_j in p_i, $m_{i,j,k}$ is the possibility of mapping a_j in p_i to e_k, and $s_{i,j}$ is the processing workload size of a_j. In this paper, we assume that $s_{i,j}$ is proportional to the input file size. If the mapping to the executor is possible, $m_{i,j,k}$ is equal to 1, otherwise it is equal to 0. When the EPC is large, it means that we guess the performance of the executor will be high. Note that we can treat both the potential capability and the load of executors with the EPC because we calculate the EPC by using execution history dynamically.

To distribute activity instances to top N executors, the thresholds of a processing load size set up by dividing the area into N areas and the executors of top N EPC are allocated activities' instances from the size of a processing load size, based on those thresholds. As methods for dividing the area, size of the area can be equal or proportional to the EPC, integer progression, or others. We choose the integer progression division based on a preliminary experiment, by setting the thresholds ($w_r, 1 \leq r \leq N-1$) are as follows:

$$w_r = (\sum_{x=1}^{r} x / \sum_{y=1}^{N} y) \cdot S_{max}$$

where S_{max} is maximum load size of the activity instance in the execution history.

3.2. Executor Failures

The reasons of delay of the response from the executors are network congestion and failure of network or executors. When the reason is network congestion, it is already considered by EPC because the processing is just performed late. However, when the reason is failure of network or executors, the processing will be stopped. Therefore we introduce the time-out concept to correspond to the failure.

In the OXTHAS re-scheduling strategy, when time-out is occurred at one executor, workflow engine re-allocates activity instance using OXTHAS-N for other executors again. If the relationship between N of OXTHAS-N and N_{exe} is such that the number of executors that have not been allocated that activity instance is $N_{exe} < N$, OXTHAS-N_{exe} is used instead.

3.3. Adjustment Methods

As described previously, when the time-out concept is introduced, it is necessary to decrease the effect of failure of

executors. Accordingly we propose two adjustment methods: one is named " EPC adjustment method", and the other is "Time-out adjustment method". The definitions are as follows:

EPC adjustment method

We re-calculate the EPC ($c'_{k,j}$) by considering the frequency of occurrence of time-outs at each executor.

$$c'_{k,j} = \frac{c_{k,j}}{1 + \alpha \cdot R_{exe}}$$

where R_{exe} is the number of time-outs in each executor and α is constant.

When the time-out is occurred, that EPC becomes small and the unreliable executor is not allocated the activity instances.

Time-out adjustment method

We define the time-out value (T_{exe}) considering the frequency of occurrence of time-outs at each executor.

$$T_{exe} = T_{init} - \beta \cdot R_{exe}$$

where T_{init} is initial time-out value and β is constant.

When the time-out is occurred, that time-out value becomes small. Thereby the time-out is occurred soon even if the unreliable executor is allocated the activity instances.

In the adjustment methods, the more unreliable the executor is, the larger penalty it suffers. If the workflow engine receives an activity instance from the executor that may be crashed after the time-out occurred in that executor, EPC is set to $c_{k,j}$ from $c'_{k,j}$ and time-out value is restore to T_{init}, because that executor is not failed nor recovered. Therefore the adjustment methods can also correspond to temporary failure of executors.

4. Experimental Validation

4.1. Simulation Description

In order to show the validity of our failure-aware scheduling method, we have implemented simulation system based on workflow management architecture in Sect.2.1. We have simulated the architecture of the WFMS on one computer. First, we vary the value of time-out value and compare the obtained results of OXTHAS-N (N=3) with different adjustment methods in the case of one executor having crashed. Next we verify the effect of the adjustment methods. We configure $\alpha = 1$, $\beta = Tinit/100$.

Since the EPC is computed on the base of execution histories, we used a random algorithm for the first 1000 steps, made measurements 10 times respectively and computed

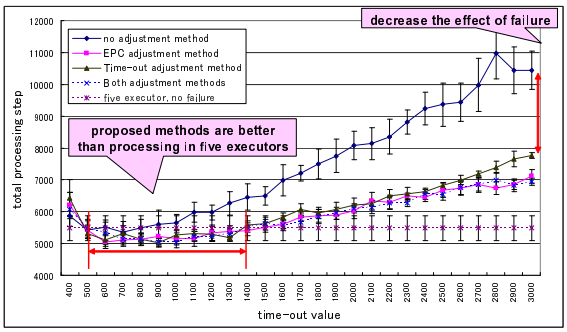

Figure 1. Total processing step of each adjustment method for time-out step setting

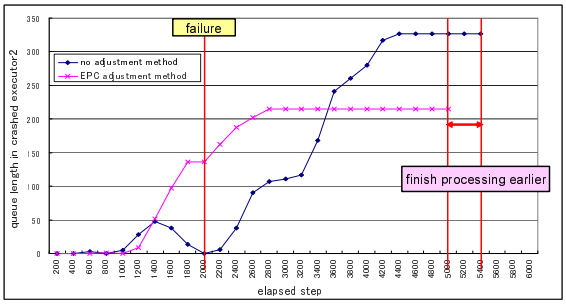

Figure 2. Queue length transition of the crashed executor

their average. The error margin stick at each graph of simulation results shows two-sided 95 percent confidence interval.

We set the number of executors to six and set the number of workflow engines to two. Within each workflow engine, the same number of process instances are processed. The information about where each process instance is being processed is only known to the workflow engine responsible of the process instance enactment. As for the considered business processes, we made simple processes without branching. As for the processing performances of the executors, the size of a process instances, activity instances and so forth, they are determined at random. Moreover, the number of activity instances for all the process instances was set to five. Finally, all the executors can carry out all kind of activity instances and we only consider the complete failure for executors.

4.2. Simulation Results

In Fig.1, we vary the value of the time-out when executor2 is crashed at 2000 step and we measure the total processing step for the OXTHAS with different adjustment methods: no adjustment method, EPC adjustment method, Time-out adjustment method and both two methods. Moreover, we compare the case of six executors including an unreliable executor with the case of reliable five executors without executor2. In the obtained results, proposed adjustment methods is better than no adjustment method. At

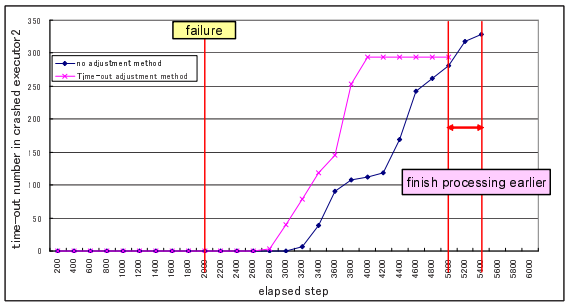

Figure 3. Time-out number transition of the crashed executor

the proper time-out step, the case of six executors including unreliable executor with proposed adjustment methods is better than the case of five reliable executors with no adjustment method.

Figure 2 shows the effectiveness of the EPC adjustment method. For the OXTHAS with the EPC adjustment method, the increase of queue length for executor2 is stopped earlier than for OXTHAS with no adjustment method. By using this method, crashed executor2 is not allocated as many activity instances as possible.

In Fig.3, we can understand why the Time-out adjustment method is effective. For the OXTHAS with the Time-out adjustment method, the increase of the number of time-outs at executor2 is stopped earlier than for the OXTHAS with no adjustment method. By using this method, if some activity instances are allocated to the crashed executor, the time-out step becomes short and they are re-allocated to the other executors earlier.

5. Related Work

In [3], authors proposed a load-balancing technique that reduces the waiting time before the workflow engine starts processing the process instance by deciding to which workflow engine a certain process instance is allocated. In their approach, authors studied a distributed WFMS architecture with distributed worklist management mechanism and load balancing sub system. Although this load index has realized load-balancing between workflow engines, when we consider the case that multiple workflow engines share the executors, it is important to consider load balancing between executors. But this research does not deal with load balancing between executors.

Nonobe and Ibaraki have proposed an algorithm for scheduling with resources restrictions [7]. It considers work assignment in a closed WFMS where there is only one workflow engine. The shortest work to be processed is assigned first. however, if we consider the case where there are two or more workflow engines, if each workflow engine is only aware of its assigned work, in case of network changes , this algorithm cannot be used as it is.

[6] has proposed highly available and reliable distributed execution of web-service based workflow management with distributed control. But this research does not deal with web-service based workflow management with centralized control.

6. Conclusions

In this paper, we proposed a re-scheduling strategy for load-balancing and failure handling for centralized Web-services based workflow management. Specifically, we have explained our model of Web-services based workflow management and some characteristics of that model. We have explained the OXTHAS scheduling strategy to balance executors' load. In our proposal strategy, we proposed two adjustment methods using the time-out concept to correspond to the executor and network failure. Then we evaluated the effect of our proposal methods by implementing a simulation system and showed that the methods are able to hold down the effect of the failure and also able to process the activities' instances efficiently even if the unreliable executor is used.

As future directions, we intend to take into the consideration the failure that cannot process only a part of activities. We may also implement effectively a WFMS and confront the results with the simulator results.

Acknowledgments

This work is partially supported by MEXT of Japanese Government via Grant-in-Aid for Scientific Research #16016232, the JST CREST and the Tokyo Tech 21st Century COE Program.

References

[1] BPEL4WS. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[2] Workflow Management Coalition. http://www.wfmc.org/.

[3] L. jie Jin, F. Casati, M. Sayal, and M.-C. Shan. Load balancing in distributed workflow management system. In *SAC2001*, 8 2001.

[4] H. Katoh, T. Kobayashi, and H. Yokota. Simulation evaluation of a load balancing method for workflow management with web services (in japanese). *DBSJ Letters*, 4(2):25–28, 9 2005.

[5] H. Katoh, N. B. Lakhal, T. Kobayashi, and H. Yokota. Oxthas: A method for balancing loads in workflow management systems with web services. Technical Report 06-0005, Tokyo Institute of Technology, 3 2006.

[6] N. B. Lakhal, T. Kobayashi, and H. Yokota. Throws: An architecture for highly available distributed execution of web services compositions. In *Proc. of RIDE 2004*, pages 103–110, Mar 2004.

[7] K. Nonobe and T. Ibaraki. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 557–588. Kluwer Academic Publishers, 2002.