SERVICE SCHEDULING IN THE WOS(TM)

STEFAN SCHWETSCHKE, MATHIAS STEIN, THOMAS BÖHME
AND HERWIG UNGER

Abstract
The paper is devoted to the resource management in the Web operating System. A load sharing method is sketched. The method is based on a prediction for the resource requirements of a process.

1 INTRODUCTION

One of the basic features of the WOS is that a WOS-node may use a service on another node. In order to achieve a nearly even utilization of the nodes, a kind of load sharing method is needed. Since the WOS is organized in a strictly decentralized way the load sharing method should not involve any kind of central installations. Moreover, it should not need much additional communication either (see [5, 3]).

Our approach works as follows: When a WOS-node (called the client) wants a service to be executed on another WOS-node (called the server), it invokes a warehouse to obtain a list of servers on which the requested service is available. (If this warehouse does not "know" a suitable server, other warehouses will be searched for one.) Next a service request is sent to a server from this list. The server then decides whether the service can be executed according to the parameters specified in the request or not. If yes, the service will be accepted. Otherwise, a message is sent back to the client. In this case the client sends a service request to the next server from the list, and so on. The server’s respond is stored in the warehouse and will be utilized to organize the servers on the list in an appropriate way when the next request is received by the warehouse (see [6]).

Specifically, two kinds of services will be distinguished: Services that do not involve any kind of interaction (noninteractive services) and interactive services. Non-interactive services will be associated with a deadline for the execution of the service. For interactive services a response time is specified. Thus a service request consists of list of parameters that includes the name of the service to be executed, the input file, the starting time, i.e. the earliest date at which the execution of the service may be started, the deadline for the execution of the service if the service is noninteractive and a responds time if the service is interactive. Additionally, the service request may contain some other optional parameters. In order to decide whether a service request can be accepted, it has to be checked whether the service can be executed according to the conditions specified in the request without disturbing the execution of the already accepted service requests. Obviously, the proposed approach requires a prediction for the resource requirements of the process executing the requested service as well as for the load of the server, prior to the execution of the service.

The paper is organized as follows. The following two sections are devoted to the discussion of the prediction unit and the scheduler, respectively. In the last section some remarks about
future work and related problems are made.

This paper reports about an ongoing research. Hence we mainly discuss problems and possible approaches for their solution; rather than presenting final solutions.

2 PREDICTION OF RESOURCE REQUIREMENTS

This section deals with the prediction of the resource requirements of a process executing a requested service.

Specifically, the CPU time, the file I/O and the memory requirements of the process are considered. Clearly, these three parameters depend on the input file. However, there is no general procedure to extract this information from the input file. In fact, in many cases the only way to learn about the resource requirements will be to execute the program realizing the service with the specific input file.

On the other hand, the situation may change drastically if additional knowledge about the program is available, e.g. for a compiler there might be a close relationship between the length of the input file and the resource requirements caused by processing this input file. If this is the case, a statistical method (e.g. a parametric regression) can be used to obtain a prediction for the resource requirements of the process executing the service. If such a method is applicable, a statistical algorithm will be implemented on the server and the resulting data will be stored in the warehouse.

In other cases the prediction scheme described in [1] (see also [2]) might be useful. The basic idea behind this method is the observation, that if a program is successively executed several times by the same user, the resource requirements form a certain pattern. In order to utilize this method, the corresponding algorithm should be implemented on the client side and the system should allow for a transfer of the resource usage data from the server to the client.

Finally, if for a certain service no practical prediction method is available, the user of this service should be notified of this fact and the service should be treated as an interactive service, i.e. a deadline for this service must not be specified.

The response time of a process depends on its resource requirements, the architecture it is running on and on the load of the node during its live time.

In [4] a method is proposed to predict the response time of a process from its resource requirements and the load of the respective node. The method uses a fuzzy decision manager. Especially, this method could modified so to predict the appearance of swapping operations.

3 THE SCHEDULER

The scheduler holds the schedule of all accepted service requests. When a new service request arrives, it is checked whether it can be added to the schedule in such a way that none of the restrictions concerning the deadlines or response times is violated. For this purpose the execution of the schedule is simulated. Given a round robin CPU scheduling this can be easily done without causing much additional overhead:

- If during the simulation a restriction for an already accepted request is violated but the new request can be executed according to its specifications, the scheduling priority of the
newly added process is reduced; if the priority is already as small possible, the request is rejected.

- If the new request cannot be executed according to its specifications but all other requests can, the priority of the new process is increased; if it is already as large as possible, the request is rejected.

- If a restriction of the new request and a restriction of an already accepted one is violated, the new request is rejected.

- If none of these cases applies, the new service request is accepted.

Special attention has to be paid to the fact that in general also local users will work. In order to avoid unwanted influence of the WOS jobs on the local jobs it is proposed to reserve a fixed amount of CPU time for the local jobs. To realize this, a patch has to be added to the operating system.

4 THE DECISION MANAGER

In figure 1 the principal structure of the decision manager as part of a WOS-node is shown.

When a service request is received by the server (shown on the right hand side) the remote resource unit (RRCU) checks with the security system whether the request can be accepted
in principal. If yes, the scheduler is invoked. The scheduler fetches the schedule of the node and, if the requested service is noninteractive, a prediction for the resource requirements of the process to be started. The schedule of the node is taken from a local warehouse. Depending on the requested service the prediction for the resource requirements is either also contained in a local warehouse or is part of the service request. (The latter is the case if the prediction is provided by the client.) Using these data the scheduler checks whether the service can be executed according to the conditions specified in the service request or not. If yes, a process executing the service is added to the schedule. Otherwise, a message is sent back to the client.

5 CONCLUSIONS AND FUTURE WORK

It is planned to implement a decision manager based on the discussion in the paper. To keep the system platform independent the code will be mainly written in JAVA. A simplified version of the scheduler has already been implemented. Furthermore, it is planned to conduct trace driven experiments to analyze the accuracy of the proposed prediction methods.

REFERENCES


