Management of Efficient Service Provision in Distributed Systems

DIRK THIßEN
Department of Computer Science, Informatik IV
Aachen University of Technology
Ahornstr. 55, 52074 Aachen
GERMANY

Abstract: The increase in service provision over the internet as well as the trend to establish global open service markets supply a basis for business scenarios in which e-commerce applications become more and more important. To survive in the arising competition situation between different service providers, services have to be offered not only in a wide range and for a suitable price, but additionally they have to be deployed in an efficient way, promising a good quality and performance in service provision to satisfy the customers. Because existing internet infrastructures have to be integrated and used in service provision, it becomes necessary for service providers to develop new concepts for the management of the arising service-oriented distributed systems. This paper discusses mechanisms for the management of such services in distributed environments. A trader is used as a central component, supporting a customer in choosing a service and a provider in delivering this service efficiently. For this purpose, the trader is enhanced to consider as well quality aspects of a requested service, as the global state of the distributed system when selecting a service. To fulfil performance characteristics of a mediated service at usage time without the necessity to involve application developers and users, a load balancer and management proxies are integrated into the trader for a proactive management of the service usage process.

Keywords: E-Commerce, Open Service Market, Trading, Load Balancing, Management Proxy, CORBA, Distributed Systems

1 Introduction

With the globalisation of service provision due to the development of network and system technologies and the tendencies to create open service markets, new changes for offering commercial services at the customers’ desktop and for the cooperation between enterprises arise. At the same time, this evolution leads to distributed systems with a high degree of complexity and heterogeneity. A server providing a service in such a system requires a certain level of service quality in terms of e.g. performance, availability, and reliability of the components involved to satisfy its customers. But the management of these services is a complicated task, because a lot of applications, systems, and network infrastructures are involved in the process of service usage. The handling of heterogeneity becomes easier by progress in the research area of middleware platforms. In our case, as a basis for a management infrastructure, the Common Object Request Broker Architecture (CORBA) is used. But when using a middleware platform, new problems arise, because management functionality is limited in current middleware implementations. Especially in large distributed systems, where the control over applications and system and network resources is distributed over different providers, the management of services is a complicated task.

This paper presents an infrastructure supporting the efficient service provision in a global service market. The focus is on supporting the customer in service selection and manage service performance and availability. Because the violation of performance and availability aspects in this context has no disastrous consequences, like e.g. in flight control systems, only soft requirements are considered, i.e. violations of performance or availability aspects are tolerated, but tried to be avoided. To enable the flexible integration of new services and providers into the management process, users and application developers should not be involved in the management process. For this purpose, a trader-based management approach for services in distributed systems was chosen. A trader can be seen as an enhancement of a naming service, where a service is described by a service type determine the kind of the service, and service properties describing non-computational aspects of the service. Because of this structure, the trader not only is an ideal technique to support customers in searching for a service, but additionally dispo-
ses of a good knowledge base for service details necessary in the management process. Our trader is enhanced in three aspects:

- Estimate the suitability of a service for a customer by considering a customer’s quality requirements.
- Integration with a load balancer to consider the current performance situation at the service providers (and trade-off with the suitability estimation from the first aspect) for a raw proactive management by distributing requests over available resources.
- Introduction of management proxies encapsulating and grouping services for observing and keeping requirements at service usage time.

The system is designed to cause a small overhead for both, service providers and customers and to adapt to certain quality descriptions.

The paper is structured as follows. In chapter 2, a short overview of related work is given. Chapter 3 explains our distributed, trader-based approach of performance management for a distributed system. The trader modification is described and an the load balancing process as well as the management proxies are explained. Finally, chapter 4 concludes the paper and addresses some perspectives for further work.

2 Related Work

If the same service is offered by several servers, a customer will have to be supported in choosing one of them. Two powerful mechanisms are trading and load balancing. Whereas in trading the choice is driven by a customer’s requirements, the load balancing approach applies on the server side. An optimal load balancing strategy would achieve a minimal response time of the system and the servers involved. [1, 2] came to the result that simple strategies with small communication effort are most suitable, even considering more recent strategies using complex techniques like fuzzy decision theory. Strategies which were based on a threshold achieved the best results. In these strategies servers are randomly chosen for the next task if their load is lower than a threshold.

Trading is a good concept to support the binding between clients and servers in large open systems. But if one service type is searched for frequently, and many client get the same service instance, single servers or network resources can be overloaded. On the other hand, a load balancer tries to realise a perfect distribution of client requests to available servers. Yet, it can only select one server in a particular group; in large open systems, where many services with different types exist, the load balancer would have to know the type of the service to select a server. Additionally, the load balancer can not consider service properties when selecting a server. Hence, a combination of trader and load balancer seems to be a suitable solution for a load-oriented assignment of clients to servers, and thus for a proactive management of services in a distributed environment.

Although both, trading and load balancing, are current research topics in distributed systems, only little work has gone into combining these approaches. The first were [6] and [9], which made simulations for the usage of a fair service selection strategy which considers global system aspects when mediating a service. Such a strategy does not guarantee an optimal selection for each client, but tries to optimise the global behaviour of a system. They confirm the former results on the advantages of simple load balancing strategies. A main topic in [9] is the use of the knowledge of a server’s load a trader has from previous service mediations. It was shown that a cyclic assignment of clients to the servers achieves the same distribution quality as load balancing. Yet, this approach is based on knowledge of the service time for each task. An approximation of this time, considering the server performance and the service type, is not feasible in heterogeneous and open systems. Furthermore, it has to be considered that not every service utilisation is arranged by the trader; for each server, there will be a load independent from the trader’s activities.

In [3], the co-operation of a trader and a management system is discussed. A trader is implemented on top of a management system and uses its functions to request the values of dynamic attributes or a server’s load by mapping management attributes onto service properties. Disadvantages of this approach include the trader’s dependency on the management system and the lack of a special load balancing component: a load distribution is not transparent for a user, who must specify special complex optimisation criteria. In [7], an integration of load balancing for middleware platforms with an interface definition language is proposed for the distribution of load in a heterogeneous distributed system. The stub generated from the interface definition is enhanced by a sensor component. This sensor transmits load information to a local load balancer, which propagates all information to the other local balancing components to achieve an overall view of the system load. A naming service uses a local load balancer if a client requests a service. This con-
cept is less powerful than the approach proposed in this paper, because only a naming service, as opposed to a trader, is used.

3 Integration of Load Balancing into Service Trading and Provision

An infrastructure for the efficient management of services in a distributed environment has to cope with many aspects, like security, accounting and performance management. In this paper, we are only looking at the aspects of the service selection and the performance of service provision, see figure 1.

The processes of service selection and performance management are related: when a customer selects a service, it expects a good service performance. One problem in service selection arises from the dynamics of open distributed systems: service properties regarding performance aspects, which are given at service selection time, can vary at the time of service usage. Performance management then aims at planning, managing, and – in the optimal case – guaranteeing capacities of a resource. Special approaches of bandwidth reservation or internet QoS are not appropriate in many cases, because the communication infrastructure does not support such approaches, the customers don’t want to pay the extra effort, or for some service types such mechanisms would cause to much overhead. Looking at services, additionally the ‘resource’ consists not only of networks, but of systems and applications, too. Thus, one have to find a technique to manage all these aspects at once. Because a trader disposes of several information regarding as well service provider as customer, it forms a good basis for a management of services, at the same time giving a customer a good method selecting a service. To consider performance aspects, dynamic service properties could be used. But these properties only would consider certain parts of the whole performance, and not each service provider is able to collect and provide all necessary property values. Instead, because most performance problems depend on some sort of load, we decided to integrate a load balancing mechanism into the trader.

3.1 System and application load

For an efficient integration of a load balancer into a trader with minimal effects on participating clients and servers, it should be possible to combine the ordering of the services made by the customer’s constraints with the ordering of servers with respect to their load. The load distribution process has to be transparent for the user, but he should have the option to influence the process, e.g. by defining which weight the load parameter should have compared to the service quality. Furthermore, the load balancer should be flexible to enable the use of several load balancing strategies and load meanings.

The architecture of the enhanced trading system is shown in figure 2 [8]. On each server host a monitor is installed to observe all local servers. The monitor is connected to the load balancer, which is located on the trader host. The trader and the load balancer co-operate to select a service (a server) for a client request. A client imports a service from the trader and connects to the corresponding server. The implementation of this architecture bases on IONA’s CORBA implementation Orbix 2.3.

Service usage can be determined using a variety of values, e.g. the CPU load, the network load, or the load caused by i/o operations of the application, e.g. read/write operations on a database. By using the load balancer, we only considered the CPU load as a measure for the utilisation of the system as well as the application. To determine this load, the servers’ queue length, the service time, and the request arrival rate can be used. Each participating server is enhanced by a sensor, which collects these information and sends them to its monitor. These sensors are realised using CORBA interceptors. But when using inter-
ceptors, an application has to allow the respective adaptations for participating in the load balancing process. As some applications might be legacy applications, management proxies were constructed to enhance an application with the necessary functionality without the need of changes in the servers itself [5].

As load information we use the service time in real time, the service time in process time, the usable CPU performance, and the queue length. A monitor manages a local management information base of load information and enables the load balancer to access it. The monitor not only stores the received load values from the sensors resp. the proxies, but also calculates additional, more “intelligent” values. This includes the computation of a floating average value for the load values mentioned above as well as an estimation of the time to process all requests in a server’s queue.

Based on a client’s service specification, a common trader searches its service directory. This process can be interpreted as a sorting of fitting services according to the degree of meeting the client’s restriction. For the integration of a load balancer this sorting is not sufficient, because the servers’ load must also have an influence on this order. To enable the consideration of both the trader’s sorting and load aspects, the trader must assign a score characterising the degree to which the client’s requirements are met by each service offer. To obtain such a score, a method as that proposed in [4] can be used. When the trader informs the load balancer about a server, the quality score for its service offer is calculated and also passed to the load balancer. The load balancer calculates a load score for each service offer, that is, the server with the lowest score has the lowest load. Three strategies which try to minimise the system’s load regarding to a particular load metric were implemented (all are simple strategies due to the results in [1, 2]):

- **Usage_Count** counts the number of requests mediated by the trader to a server previously.
- **Queuelength** considers the current number of requests in a server’s queue.
- **Estimated_Time_to_Work** estimates the time a server has to work on all requests in its queue.

The load balancer evaluates the most suitable service offer by making a trade-off between the quality score and the corresponding server’s load score instead of sorting the result list relating to the client’s constraints. To influence the evaluation, information about the client’s weights regarding quality and load score is passed to the load balancer as well as a metric to combine both values. After all scores have been computed, a minimisation of the scores gives the ‘best’ suited service offer. The load balancer notifies the trader to inform the user about the selected service.

### 3.2 Network load and dynamic application & system load

For gathering load values for applications or systems, in section 3.1 sensors resp. management proxies were used. But the approach in section 3.1 has not considered network load, nor it is able to consider dynamic changes in the load situation after the mediation of a service, i.e. it only considers a snapshot of the system’s and servers’ states when a customer requests a service. This approach is not working well, if certain network routes are heavily loaded, or if a service usage consists of several requests over a longer time, because system or server load could increase in this time.

![Flexible Management Proxies](image)

A more promising approach enhances the role of management proxies to improve the load balancing process. As shown in figure 3, the proxy hides the details of an application from a client by encapsulating its interface and offering a new CORBA interface to clients and management applications (the load balancer in this case). In this way, a uniform interface can be offered to management applications, hence a management application is freed from adapting to different application details. The management proxy passes client requests to the original interface of the application. Internally, the wrapper contains management functionality. This concept allows a manager to obtain many detailed management information collected by the proxy. Internal functions could include for example measurements, surveying of thresholds, and statistics. In this way, the management proxies are not ‘intelligent’, but they are able to collect data and notify other components about changes. This simple usage of management proxies can be enhanced for participating in the load balancing process.
A trader only obtains information about server-specific service properties, and it is involved in a service usage only at selection time. Thus, we needed control elements closer to the network and usable at service provision. For this purpose, we enhanced the management proxies from only collecting load values to react on changes in server utilisation. By registering a management proxy at the trader instead the server itself, a customer is sending its requests to the proxy. Based on the current load situation at its server and the server host, the proxy can re-route the request to another server providing the same service, or change the priority of the server process at the host to spend a higher performance portion to the server. This approach can consider dynamic changes in application and system load, but by placing the management wrapper near the server, nothing can be done considering network load. Thus, in a further step, the proxies are detached from single servers, see figure 4.

![Diagram](image)

Fig. 4: Request distribution by management proxy

Proxies now are placed at several hosts distributed over the system (ideally near a customer) and manage a set of similar services. The similarity is given by identical service types and small deviations in the service properties. Such a set of services is determined by the trader at service mediation. A proxy is informed about this set and the client gets the proxy reference instead of a fixed server reference. Thus, a client is not directly connected to a server, but to a server group. The proxy observes as well the network as the system load (as for the load balancer) for its servers. Thus, by this approach dynamic changes as well as network load can be considered when assigning a server to a client request. The influence of network and system load on the choice of a server depends on the service type. E.g. if a request is very small but needs complex computations on server side, the system load must be considered with priority, but if lot of data are to be transmitted, the significance of network load increases. Thus, when designing the system, criteria for each service type have to be defined.

For the design of the management proxies, we had to make some decisions. First, a service usage can consist of a single call, or of a sequence of calls. In the latter case, all requests of one client have to be sent to the same server. Thus, the management proxy have to generate a table, in which it holds relations between clients and selected servers. For the first evaluations, we decided only to consider service usages consisting of one call. Second, the grouping of services can be made by similarity of service definitions, or by fulfilling a client’s request. For the latter case, a management proxy would have to manage different server tables for each requesting client, thus we decided to use the similarity criterion. Furthermore, each client contacts a management proxy, which connects it to a server. As an enhancement, chains of management proxies could be used for larger systems, in which the management proxies only know parts of the whole service market.

It must be taken into consideration that the introduction of the management proxies adds an overhead to the underlying system regarding service execution times. Although this is expected to be comparatively small, it has to be examined in the context of real time applications.

### 3.3 Performance of trader and proxies

To show the suitability of the approach presented, we have made some measurements in a distributed environment, where the same service is provided by several instances.

![Graph](image)

Fig. 5: Trader with load balancing abilities

The performance of the servers is valuated by looking at the response time, i.e. the time a server needs to work on a request and send back a response. From figure 5, it can be seen, that the dynamic load balancing strategies Queue-length (QL) and Estimated_Time_to_Work (ETTW) can clearly improve the response time compared to a systems without load balancing, especially in situations of high load, i.e. many client requests. On the other hand, if placing a management proxy between client and server, a computation over-
head is caused. This overhead depends on the proxy’s functionality and the size of the request which has to be analysed, see figure 6.

Fig. 6: Overhead caused by management proxies

From figure 6 it can be seen, that even for large request of 100 KB, only a small overhead is added to a server’s response time. For most scenarios, this overhead is smaller than the advantage obtained by introducing the trader’s load balancing. Overall, it can be said, that the performance of service provision can be enhanced by the combined load balancing concept.

4 Conclusions
To ease the selection of a service provider in a global service market, a customer can use a trader for service selection. But common traders are not able to deal with QoS issues at service usage time. To enhance a trader with such abilities, a load balancer was integrated into the selection process. This enhancement allows the trader to consider application and system performance in the trading process. To consider network load and dynamic changes in all load aspects, management proxies encapsulate server groups and can proactively react to system changes.

Measurements have shown, that the consideration of a load balancer can improve a server’s answer time up to more than 50 percent. On the other hand, when using a management proxy to pass a request to a server, an execution overhead have to be tolerated. This can worsen the response time in cases, where the service usage would be very short, but assuming, that a service would have to make database operations etc., this case would be very seldom.

While the integration of management proxies seems to be a promising approach for non-time-critical application areas, there remain some issues for further work. First, we want to look, if there are other ways of integrating the network load directly into the trader by co-operating with a configuration manager. Furthermore, we have to examine the issues of proxy design not considered as far, i.e. related client requests, grouping of services oriented at clients and chains of proxies. We have to make more measurements considering different network load situations and the effect onto proxy computation overhead and improvement of service response time. Last but not least, by grouping not only several service instances of one provider, but similar service instances of different providers, problems in accounting of a service could arise.

References: