Managing Services in Distributed Systems by Integrating Trading and Load Balancing

Dirk Thißen
Aachen University of Technology,
Department of Computer Science, Informatik IV
Ahornstraße 55, D-52074 Aachen, Germany
Phone: +49-241-8021410, Fax: +49-241-8888220
thissen@i4.informatik.rwth-aachen.de

Helmut Neukirchen
Medical University of Lübeck,
Institute for Telematics
Ratzeburger Allee 160, D-23538 Lübeck, Germany
Phone: +49-451-5004867, Fax: +49-451-5003722
neukirchen@itm.mu-luebeck.de

Abstract

With a changing structure of networks and application systems due to the requirements of decentralised enterprises and open service markets, distributed systems with rapidly increasing complexity are evolving. New concepts for an efficient management of such systems have to be developed. Focussing on the service level, examples for existing concepts are trading to find services in a distributed environment, and load balancing to avoid performance bottlenecks in service provision. This paper discusses the integration of a load balancer into a trader to adapt the allocation of client requests to suitable servers due to the current system usage, and thus to improve the quality of the services in terms of performance. The approach used is independent of the servers' characteristics, because for the servers involved, no provision of additional service properties to cover load aspects is necessary. Furthermore, it is flexible to enhance, because the concept of load used can be varied without modification of trader or load balancer.

1. Introduction

The integration of small isolated networks into bigger ones, distributed over the whole world, caused a change in the design of application systems. Single applications are replaced by distributed ones, which can utilise resources more efficient. By using the concept of a middleware, e.g. the *Common Object Request Broker Architecture (CORBA)*, it is possible to create an open service market, in which services provided by different application objects can be used to compose the functionality of a new application. One example for the use of this concept is given by the *Cooperative Research Centre 476 IMPROVE* [8], a project at Aachen University of Technology we are participating in, which aims to support chemical engineers in designing chemical processes by support of computer science. The chemical engineers are located in different parts of a company, or even in different companies. To provide support of single engineers as well as the whole design process, a distributed design environment of the tools and services involved has to be realised. A trader is used to find applications providing the demanded services in this environment. To enable an efficient work on a design task, it is possible to distribute requests among several applications providing the same service. Nevertheless, a system can by partly overloaded, if one server is used intensively.

This paper addresses the enhancement of a trader by a load balancing component. Thus, the trader is enabled to consider performance aspects when selecting a server for a client's request. The trader has to search for a service which is optimal in two senses. First, the service quality determined by service properties has to have an influence. Second, the load of the corresponding server has to be considered. From these factors, a compromise has to be effected. This concept was implemented and evaluated on a CORBA basis. The paper is structured as follows. In chapter two, the concepts of trading, load balancing, and a combination of both are explained. Chapter three introduces our approach for such a combination. In chapter four, some examinations for showing the usability of our approach are given. Finally, chapter five concludes the paper and addresses some perspectives for further work.

2. Trading and load balancing

If the same service is offered by several servers, a client can be supported in choosing one of them. Two powerful mechanisms for doing so are *trading* and *load balancing*. Whereas in trading the choice is driven by a client's demands, the load balancing approach applies on the server's side.

Trading: The trading service can be seen as an enhancement of the naming service, which gives a client more flexibility in specifying the service it needs [10]. Whereas at the naming service a server resp. the service this server

offers must be assigned with a unique name, in the trading concept a service is described by a service type and service properties. The *service type* describes the functionality of a service and determines its signature or interface definition. Thus, an open service market is enabled. But the service type is not enough to describe a particular service. Additionally, *service properties* can be used to describe non-computational aspects of a service. Generally, service properties are divided into two groups: *static* and *dynamic properties*. The group of static properties contains all service properties, which do not change over time. The dynamic service properties have varying values and can be used as performance measurements. Using service properties, a server describes its capabilities in service delivery. A client searching for a service, can formulate as well restrictions on the service properties to express its needs, as criteria to determine an order on the services found, for example a minimisation of a property value. The trader matches the client's description against all recorded services and passes back a list of references for the corresponding servers. Traders were implemented in various environments [4], but they got not very popular. Only with the adoption of trading as a CORBAservice, it became of general interest. Today, a lot of implementations of traders for CORBA platforms exist.

Load balancing: Load balancing has the goal to uniformly utilise all available resources in a system by distributing tasks according to a given strategy. Simple strategies are *static*, i.e. new tasks are distributed to the servers by a fixed schema. Examples are the cyclic assignment of tasks to the given servers or random server choice. These strategies are easy to implement, but cannot adapt to special situations. The more promising strategies are adaptive to a system's state, so they can react on sudden changes in the system. Tasks are distributed according to the load of the available servers. Such d*ynamic load balancing strategies* have to consider the fact, that measured load values only reflect the past. By updating the load values more often, this problem can be minimised, but the network load for transmitting load information increases to much. A compromise between communication overhead and relevance of the data has to be made.

A lot of work on load balancing exist for mostly homogeneous systems. As an example, [3] came to the result, that simple strategies with small communication effort are most suitable. The best results were achieved with strategies basing on a threshold. Servers are randomly chosen to get the next task while their load does not exceed the threshold value. More current strategies use more complex techniques, for example fuzzy decision theory [2]. Such work contradicts the advantage of simple strategies. On the other hand, [1] affirms the former results on simple strategies.

Combining trader and load balancer: Trading is a good concept to support the binding between clients and servers. But if one service type is searched for very often and each client wants to get the "best" service instance, single servers can be overloaded. On the other hand, a load balancer tries to realise a perfect distribution of the clients' requests to the available servers. But it only could select one server in a particular group; in large open systems, where lot of services with different types exist, the load balancer would have to know the type of the service, for that he can choose a server. Additionally, the load balancer only could choose a server by its load, not by other service attributes. Thus, a combination of trader and load balancer seems to be a suitable solution for a load-oriented assignment of clients to servers in a distributed system. The direct approach would be the usage of *load values as dynamic service properties*. The trader could make a load distribution by means of these attributes. But for a service provider, it could be hard or even impossible to provide an interface where the trader can request the information for dynamic attributes, especially in cases, where legacy applications are used. Furthermore, this concept is inflexible, because an enhancement of the meaning of the load value by other load information aspects would be hard.

Although both, trading and load balancing, are current research topics in distributed systems, only few work exist in combining them. One example is [12], which made simulations for the usage of a "social" service selection strategy. Such a strategy does not guarantee an optimal selection for each client, but tries to optimise the global behaviour of a system. This work affirm the results of the former work on load balancing regarding the usage of simple strategies. Additionally, the risk of an oscillating overload of single servers is mentioned. As a solution, a dynamic strategy with a random component is proposed. A main topic in [12] is the usage of the trader's knowledge about former service mediation. It was shown, that a cyclic assignment of clients to the servers and using load balancing, achieve a similar load distribution. But this approach bases on knowledge about the service time for each task. An approximation for this time by considering the server performance and the service type is not feasible because of the heterogeneity and the openness of real systems. Furthermore, it has to be considered, that not each service utilisation is arranged by the trader; on each host, there will be a load independent from the trader's activities.

In [5], the co-operation of a trader and a management system is discussed. A trader is implemented on top of the management system and uses its functions to request the values of dynamic attributes or a server's load. Dynamic attributes are obtained by mapping management attributes onto service properties. By specifying complex selection and optimisation criteria for a service selection, a load distribution can be made. Disadvantages of this approach are the dependency of the management system and the

lack of a special load balancing component. Furthermore, a load distribution is not made transparent for a user, but must be made by him with special optimisation criteria. In [11], for the distribution of load in a heterogeneous distributed system, an integration of load balancing for middleware platforms with an interface definition language is proposed. 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 a global sight onto the load. A name service uses a local load balancer if a client requests a service. This concept has some disadvantages, too. Not a trader, but only a name service is used here, which is a less powerful approach. The stub manipulation is insufficient, because the source code must be known and no transparent integration into servers is given.

3. Architecture of the enhanced trading system

For our enhancement of a trader with a load balancing mechanism, we specified some design issues [9]. It must be possible to *use the trader without load distribution*, as well as to *combine the ordering of the services made by the client's constraints with the ordering of servers regarding to their load*. The load distribution process has to be *transparent for the user*, but the user should have the possibility to *affect the process* by special service properties. Such properties could regard the information, which influence the load parameter should have compared to the service quality. The load balancer should be integrated into the trader to achieve a *synergy effect* by exchanging knowledge between trader and load balancer. Furthermore, the load balancer should be flexible in a way, that *several load balancing strategies and load meanings* can be used.

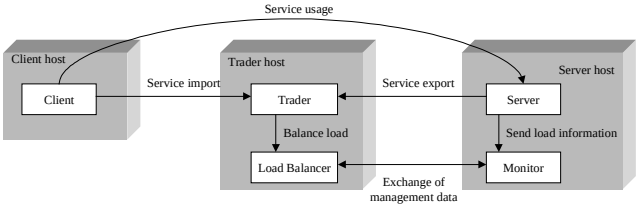

Figure 1. The trader - load balancer system

The architecture of the enhanced trading system is shown in figure 1. On each host a monitor is installed to maintain the hosted servers. The monitor is connected to the load balancer, which is placed on the same host as the trader. The implementation of this architecture bases on a trader implemented in our department using IONA's middleware platform Orbix. In the following, the components of the architecture are described.

Sensor: The load of a service usage can be determined by a lot of metrics, e.g. the CPU load, the network load, or the load caused by i/o operations. For the beginning, we only considered the CPU load. For determining the 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 gathers these load information and sends them to the monitor. Because most applications used in our scenario are legacy applications, management proxies are constructed to enhance an application with the needed sensor functionality [7]. 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. The load information is passed to the monitor as a struct as shown in figure 2. This format is used to transfer load information in the whole system.

```
interface loadbalancing_types {
  enum LoadmetricType {
                SERVICETIME_REALTIME,
                SERVICETIME_PROCESSTIME,
                PROCESSTIME_REALTIME_RATIO,
                QUEUELENGTH, ESTIMATED_TIME_TO_WORK,
                ON_IDLE, REQUEST_RATE, USAGE_COUNT,
                HOST_LOAD, UNVALID };

  struct LoadType {LoadmetricType loadmetric;
                   float          loadvalue };
};
```

Figure 2. Structure for covering load information

Monitor: A monitor manages a local management information base and enables the load balancer to access it. It manages a list of all hosted servers together with their load. Because the usage of different load metrics should be possible, all load information, which is transmitted by a sensor, is stored. Additionally, the monitor calculates more "intelligent" values. This comprise a floating average for the load values mentioned above, as well as an estimation of the time to work on all requests in a server's queue by using the mean service time of the past service usages and the past time of the current request to estimate the time the server has to work on all of its current requests. The monitor can use a caching or a polling strategy to update the load balancer's information. Based on the access and change rates for load values, a dynamic switch between caching and polling is possible. This mechanism is shared by load balancer and monitor. In case of using the polling strategy, the monitor has both information, thus it can switch to the caching mechanism. On the other hand, if caching is used, the load balancer knows about access and change rates, thus it can switch to the polling mechanism.

Trader: Basing on a client's specification, the trader searches its service directory. Services fulfilling the specification are stored in a result list. At this point, a modification of the trader was made. In the normal case, the trader sorts the result list regarding to the client's constraints. This process can be interpreted as a sorting of the services regarding to their fulfilment of the client's

quality demands. This is no more sufficient, because the servers' load must have an influence on this order, too. Thus, each service is assigned with a score characterising the degree of the client's quality fulfilment. To obtain such a score, a method as proposed in [6] can be used. When a new entry into the result list is made, the trader informs the load balancer about the corresponding server and its quality score. The trader gets back the resorted list from the load balancer and passes it to the client. In addition to the load balancer's mechanisms, the trader implements a *random strategy* to determine an order on the services found. This can be seen as a static load balancing strategy.

Load balancer: The load balancer manages two tables: one table contains the load information for the servers, the other one records each service offer found by the trader together with the trader's quality score. The load balancing approach chosen here consists of two steps. First, by using a load balancing strategy, a valuation of the services found regarding to their load as recorded in the first table is made. Based on the load values the monitors capture, three strategies which try to minimise the system load regarding to a particular load metric were implemented:

- *Usage_Count (UC)* only counts the number of requests mediated by the trader to a server in the past.
- *Queuelength (QL)* considers the current number of requests in a server's queue.
- *Estimated_Time_to_Work (ETTW)* calculates the estimated time a server has to work on the requests currently in its queue.

The load value given by theses strategies is seen as a score for a server, meaning that the server with the lowest score has the lowest load. This value is recorded in the second list. The second step is combining the score given by the load balancer with the quality score calculated by the trader. For doing so, the manhattan metric resp. the euclidean metric are used to calculate an overall valuation for each service offer. The client can influence this combination by specifying weights for quality and load score.

4. Evaluating the enhanced trading approach

To evaluate our approach, measurements were made to see the effects on server utilisation, response times and service selection time.

Environment: To evaluate the enhanced trading system, a scenario is needed, which resembles a real system's usage. Thus, a request sequence was generated to approximate a real scenario. One restriction was the avoidance of a system overload, but temporary overload situations are desirable. Thus, our request sequence contains request bursts and intervals of silence. The service time for the requests was varied from 1.1 to 16 seconds. A multi-threaded client was used to send all requests to the trader and to use the selected server directly. For load balancing, the strategies *Random*, *UC*, *QL* and *ETTW* as described above were used. To evaluate the new component, the mean response time of the servers as well as the service selection time the trader needs were measured at the client. The measurements were made in a local network (10/100 Mb/s-Ethernet). For examinations in a homogeneous system, four Sun UltraSPARCs with 167 MHz and 128 MB ram were used. To make examinations in a heterogeneous environment, four different Suns with between 110 and 167 MHz and between 32 and 128 MB ram were used. In all measurements, a restriction to a caching strategy to avoid communication overhead was made.

In the first three measurements, the benefit of the implemented load balancing strategies was evaluated. In these examinations, only the system configuration and the requested service type were varied. The order, the trader determined by the fulfilment of service properties was neglected. In the forth measurement, additionally the trader's ordering was considered.

Homogeneous server performance: At first, a homogeneous system with four equally equipped servers was used. All requests had the same service time of 1.1 second. In figure 3, the mean response times of the servers are shown. The strategies UC, QL and ETTW caused very similar response times, and for load situations of less than 50% they nearly achieved the optimum of 1.1 second. For higher load situations, the results hardly deteriorate. On the other hand, the random distribution of requests to servers dramatically impairs. For the given scenario, UC has the best performance, because all servers need the same time to process one request. For a system with homogeneous computer performance and no server usage without contacting the trader, a good load balancing is possible only with the trader's knowledge. Furthermore, it can be seen, that for load amounts exceeding 40%, caused by the constant service time and the homogeneous server performance, the more complex ETTW strategy is slightly better than the simple QL strategy.

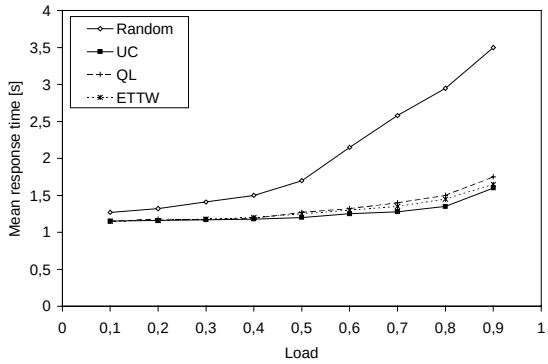

Figure 3: Homogeneous system

Heterogeneous server performance: Another behavior of the load balancing strategies is given, if a heterogeneous environment is used. As a result of the different computer performances, the service times were varied between 1.1 and 2.2 seconds. The results are shown in figure 4.

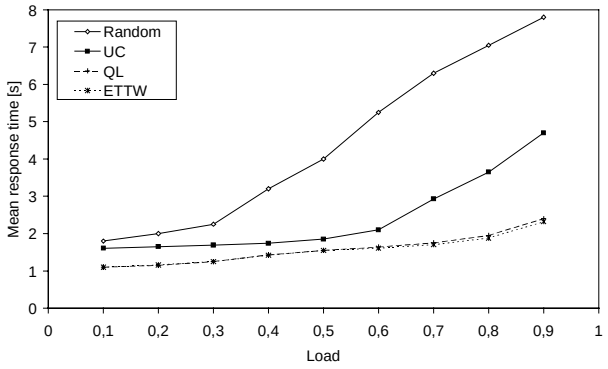

Figure 4: Heterogeneous system

In this case, the random strategy affects the worst behaviour, too. But in contrast to the homogeneous case, the UC strategy effects a definite worse distribution than the dynamic strategies. This behaviour is as expected, because by using UC, servers with less performance get the same number of requests than more powerful servers. The dynamic strategies achieve equal request distributions, with the result, that even in situations of high load the mean response time for both strategies is only as high as the service time on the slowest server. To improve the quality of the static strategies, it would be possible to weight the distribution with regard to the servers' performance. But for doing so, the trader has to know the performance characteristics of all servers in the system, and this might be impossible in a real open environment.

Different request classes: A more interesting case for a real environment is given by considering different service request classes, i.e. classes with different service times, in combination with a heterogeneous system. In our examination, we used four request classes with service times between 0.4 and 8 seconds. The requests were randomly chosen from these classes. This examination covered two situations. First, the request classes can be seen as requests for different service types with different service times. Second, it can be seen as requests for the same service type, but the service time is temporary delayed by a background load on the server nodes. The mean response time for all request classes can be seen in figure 5.

In this examination, the static and the dynamic strategies are distinct separated. While the random strategy as expected achieved the worst request distribution, the results of the other strategies are reverse as in the first examination. UC is nearly as useless as the random strategy, because to much factors influence the response time of a server, but none of them is considered. Because the service times vary heavily, it can come to an error in the estimation of ETTW, which causes a wrong decision for the next request distributions. This is also the reason, why ETTW is worse than QL. QL only counts the number of outstanding requests; in this case, the renouncement on more information about the requests is better than the usage of potentially wrong information.

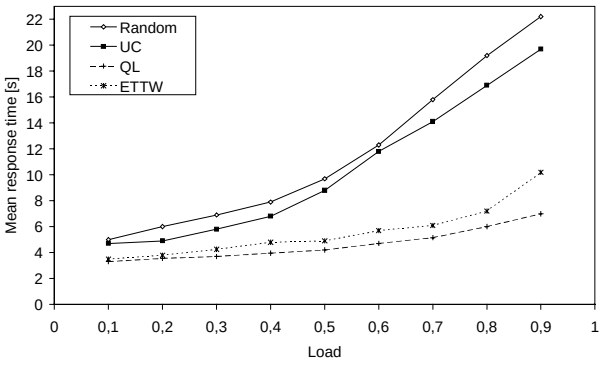

Figure 5: Different request classes

Using a compromise: In the former examinations, the trader's order on the service offers based on a client's constraints was ignored to examine only the benefit of the load balancing strategies. Because in a real situation this order denotes the user's demands on a service, in this examination both, the service quality described by the trader's ordering, and the server load were considered. Exemplarily the case with homogeneous server performance is presented. The trader's valuation of the services regarding to their quality varied between 0.25 and 1, a lower value meaning a better quality. For combining the load balancer's and the trader's ranking, the euclidean metric was used.

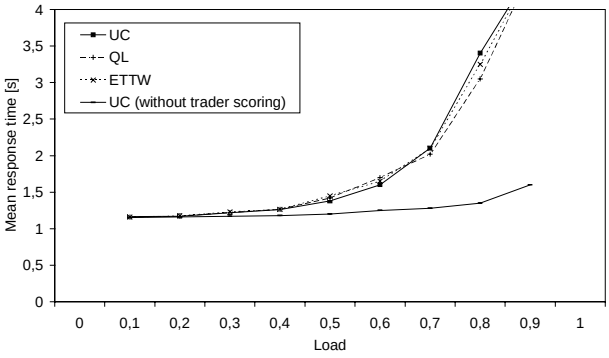

Figure 6: Considering load and quality ordering

The behaviour of the strategies is similar to the case without considering the trader's ranking, only UC is slightly worse. In the case of weighting the load with 75% and the trader's ranking with 25%, nearly no difference to

figure 3 can be seen; the assignment is made mostly by the load situation. In case of exchanging the weights, the load is nearly unconsidered, and for load situations of more than 20% the mean response time increases heavily, because nearly all requests are sent to the server which is valued best by the trader. The best result was produced by a weighting of 50% for both, load and service quality, see figure 5. By considering the service quality, the ascend in the mean response time is clearly stronger as in the case considering only the load. Especially for situations of more than 60% system load, the response times increase very strong.

Further examinations: In further examinations, we measured the influence on the trader's mediation time caused by the load balancer [9]. Generally, the time for service mediation only increases slightly, because most of the load balancer's work can be done independently from the trader.

5. Conclusions

In this paper, a combination of a trader and a load balancer was presented. Instead of using the trader's dynamic service properties, a load balancer was added to the trader as an additional component. Thus the approach is independent of the servers' characteristics and flexible to enhance. Several load balancing strategies and a concept for combining the trader's and the load balancer's results were implemented on a CORBA basis. The implementation was evaluated in several scenarios.

The optimisation of service selection regarding the servers' load seems to be a worthwhile enhancement of the trader. The response times of servers offering a service which is available in several places can be significantly reduced. The cost for this advantage is a longer time for service selection, but this overhead is very small. The usage of trader-internal knowledge about a server is useful only in homogeneous systems with low background load. For large open systems, dynamic strategies are more suitable. A simple strategy like the trader's queuelength is the best for most situations. The weighting of the load influence in comparison to the service quality at last must be a choice of the user, but the best solution seems to be an equal consideration of both.

The next step is the realisation of load balancing in a larger system, in which trader federations are used. The ascertainment of load information and the co-ordinated interworking of traders in the load balancing process give new tasks. Another extension of our approach would be the usage of methods in fuzzy logic or artificial intelligence to combine the results of trader and load balancer. The benefit of such more complex methods would be to examine. Furthermore, a usage of the load balancer component for other existing traders is interesting. In such a case, the load balancer would need to encapsulate the trader.

6. Acknowledgement

We acknowledge support from the German Research Community under SFB 476.

7. References

[1] Delicia, T.: *Modelling of Some Plain Load Distribution Strategies for Jobs in a Multicomputer System*. Informatics and Computer Science, Vol. 97, No. 1/2, Elsevier/North-Holland, 1997.

[2] Dierkes, S.: *Load Balancing with a Fuzzy-Decision Algorithm*. Informatics and Computer Science, Vol. 97, No. 1/2, Elsevier/North-Holland, 1997.

[3] Eager, D. L.; Lazowska, E. D.; Zahorjan, J.: *Adaptive Load Sharing in Homogeneous Distributed Systems*. IEEE Transactions on Software Engineering, Vol. 12, No. 5, 1986.

[4] Keller, L.: *From Name-Server to the Trader: an Overview about Trading in Distributed Systems* (in German). Praxis der Informationsverarbeitung und Kommunikation, Vol. 16, Saur-Verlag, München, 1993.

[5] Kovacs, E.; Wirag, S.: *Trading and Distributed Application Management: An Integrated Approach*. Proc. 5th International Workshop on Distributed Systems: Operation and Management, Toulouse, 1994.

[6] Linnhoff-Popien, C.; Thißen, D.: *Integrating QoS Restrictions into the Process of Service Selection*. Proc. 5th International Workshop on Quality of Service, New York, 1997.

[7] Lipperts, S.; Thißen, D.: *CORBA Wrappers for A-posteriori Management*. Proc. 2nd International Working Conference on Distributed Applications and Interoperable Systems, Helsinki, 1999.

[8] Nagl, M.; Westfechtel, B.: *Integration of development systems in engineer applications* (in German). Springer, 1999.

[9] Neukirchen, H.: *Optimising the Set of Selected Services in a CORBA Trader by Integrating Dynamic Load Balancing* (in German). Diploma thesis at the Department of Computer Science, Informatik IV, Aachen University of Technology, 1999.

[10] Popien, C.; Schürmann, G.; Weiß, K.-H.: *Distributed Processing in Open Systems* (in German). Teubner, 1996.

[11] Schiemann, B.: *A New Approach for Load Balancing in Heterogeneous Distributed Systems*. Proc. Workshop on Trends in Distributed Systems, Aachen, 1996.

[12] Wolisz, A.; Tschammer, V.: *Performance aspects of trading in open distributed systems*. Computer Communications, Vol. 16, Butterworth-Heinemann, 1993.