Scalable Service Deployment using Mobile Agents

Luis Bernardo and Paulo Pinto
IST - Instituto Superior Técnico, Lisboa Portugal
Inesc, R. Alves Redol, 9 P-1000 Lisboa Portugal
Phone: +351.1.3100345 Fax: +351.1.3145843
{lflb,paulo.pinto}@inesc.pt

Abstract. Very large networks with thousands of applications and millions of users pose serious problems to the current traditional technology. Moreover, if synchronised client behaviour exists then the limitations are strongly felt. Mobile agent systems can be a suitable technology to answer to these requirements if some aspects of the architecture are carefully designed. This paper proposes a co-operative mobile agent system with a very dynamic and scalable trading service. The system allows applications to deploy servers onto the network to respond to demand making them self-configurable. Clients can also use mobile agents with performance gains. Sets of simulations were performed to study the scalability of the overall system.

1 Introduction

In the current information technology era any new service involves a larger number of end users. Very soon networks with millions of users and thousands of applications will be at work. Despite of the size, which can by itself create operational problems, applications might produce synchronised peaks of traffic due to external events (e.g. lotto draws, sport brokering and interactive TV contests). These peaks can drive parts of the network to a halt if not properly handled. A scalable application implementation must adapt to the intensity and relative distribution of the client load using a dynamic set of servers.

A static approach to the problem, using a fixed number of servers and conventional traders, would lead to inefficient resource usage solutions: either the number of servers is insufficient (during peaks of the load) or there is an over-dimensioning of the servers deployed. A scalable world-wide application implementation should also cover two other aspects: servers must be able to run on an open world-wide set of platforms (to set the maximum load limit as high as possible and to localise interactions); and the server deployment and request binding should be tuneable (to allow service specific optimisations). Most of the traditional scalable service implementations rely on a closed set of servers (WWW cache servers [1], TP monitors [3], task load balancing algorithms [9], smart clients [20], and others). Such systems rely on system load-balancing servers or on a "head" application server (which redirects the requests to other servers), introducing potential bottlenecks. Additionally, the technology must efficiently support the dynamic deployment of new servers: Factory objects and ORB implementation repositories [13] might be used, however they are not optimised to deal with the dynamic creation of new types of servers. The required

semantic resulting from all these requirements is met with the mobile agent paradigm: It provides a simple way to deploy personalised services (replica creation from an initial service provider's server) and an open ubiquitous platform where mobile autonomous agents might run.

The solution proposed in this paper is based on a mobile agent system approach and on new scalable algorithms for the trader (called here location service) and services. Servers are implemented using mobile agents, and can deploy new replicas when the demand rises, or migrate towards the location of the majority of the clients. They use the service dynamic topological information from the location service, the client load, and the overall situation of other peer servers, to control the deployment of new servers and adapt precisely to the client load. Server agents are autonomous on their control over server deployment.

Our system assumes a scenario where clients look for a precise service. Clients use the location service to resolve unique application names to single server references. The system does not support service discoveries based on characteristics (price, availability, etc.). Each application server sets an area of the network for its service to be known (server domain). The location service takes into account its own load and server domain sizes to change dynamically its configuration. As other services, the location service is implemented using mobile agents.

The rest of this paper is organised as follows. Section 2 presents a system overview, including the client implementation and location service. Server deployment control and inter-server synchronisation is discussed in section 3. Section 4 presents a study of the dynamic behaviour of servers in face of a rising client demand. Section 5 describes related work and Section 6 concludes. This paper extends a previous one [4] by examining thoroughly the mobile agent utilisation and by extending the dynamic behaviour scalability analysis ([4] restricts its analysis to the server creation algorithm).

2 System Overview

The network provides a ubiquitous platform of agent systems, in which any agent (server or client) can run. Each agent system is tied to a location server (running locally or on another nearby system), where all the interfaces of the local agents are registered. This location server is connected to others to offer a global service.

The proposed architecture does not conflict with the standardisation efforts of OMG [15], or some of the available mobile agent systems (Aglets [10], MOLE [19], Voyager [12], etc). Mobile agents are defined as autonomous objects, which may communicate with other objects or agents through a variety of interfaces [2], and are able to migrate or start agents on remote agent systems. The main differences are the location service requirements.

When a client searches for an application name, the location service helps in the binding process (the association to a server) directing it to the nearest server. If the

location server knows more than one server, it will do splitting of client traffic. If it knows that a new, and closer, server was created it will start using the new one, and propagates this information. When a client comes for resolution it will get the best answer for that moment.

The balance between the number of clients, servers and location servers acts as a general load balancing mechanism in the system. The number of application servers will vary with the number of active clients, and the number and range of each location server will vary with both the numbers of active clients and the application servers.

2.1 Clients

Clients may interact with servers using Remote Procedure Calls (RPC) from their location nodes (or other remote interaction mechanism), may migrate to the server's agent system, or do both if, for instance, the first fails. Although not essential, mobility on the client side has advantages in terms of the overall traffic on the network: it gets more asynchronous (less demanding in “peak” network usage), can support unstable connections to mobile computers more easily, and could be tailored to satisfy a set of routing constrains according to a service specific algorithm. Performance models for both approaches ([7], [19]) showed some of the combined RPC/client mobility performance advantages when filtering large amounts of data or when performing multiple interactions over low bandwidth links. However, client migration may also cause the overloading of the servers' agent system. To avoid such effect, the client agent migration destiny is selected using the remote location server. Notice that on the system described here, the impact of variable network latency is lower than on the referred models because of the server mobility, which reduces the “distance” between clients and servers.

The system provides a simple way to integrate legacy system clients, which may interact with server agents using RPC (for instance CORBA IIOP [13]).

2.2 Location Service

The location service is one of the major players for scalability. It has two different roles: at the location server domain (the set of agent systems tied to a server), it must provide detailed local information and balance requests among local available servers (for each service); at a global level, each location server must provide a scalable global trading information service (although not complete). The global load balancing is based on network proximity relations.

The necessity for fast updating during the creation of a server clone, the propagation of frequent updates due to migration, and the dynamic nature of the information in this system (based on dynamic server domains) introduce a high overhead which invalidate some of current technical solutions based on static hierarchic systems (e.g. DNS, X.500 or OMG Trading [14]). See [4] for details.

A scalable and highly mobile system must have the following characteristics: Firstly, the application names must be flat ([18] reached a similar conclusion). Secondly, the

location search path, which is now independent from the application name structure, must be performed on a step-by-step basis, through a path of location servers where each one contain routing information indexed by the application names. Thirdly, the load and characteristics of the overall system should tune this step-by-step path.

One important feature is how the location service scales to a large population. We use a mixture of meshed and hierarchic structure where location servers at each hierarchical level interact with some of the others at that level and (possibly) with one above. Higher hierarchic levels offer a broader but less detailed vision of the services available. The hierarchical structure and the scope of the mesh change dynamically according to the load of the system, and to the size of the server domains.

The server domain is service specific. For instance, a car parking service would simply advertise on the surroundings of each car park, while a popular lotto broker service would advertise on a broader range (pricing schemes could be a deterrent to artificially large domains). Clients control their search range. Due to lack of depth, or incomplete information, resolutions can fail, and a deeper search must be tried.

3 Servers

Under the envisioned conditions, server development is a delicate task, because most of the time there will be only a vague forewarn of how high the "peak" of the load will be, or when it will happen. An adaptation algorithm is required, to achieve the service requirements.

3.1 Server Deployment

Several reasons can originate the deployment of a new server (or the migration of an idle one). A new server can help solving problems related to: 1) insufficient processing resources (the overloading of the available servers); 2) insufficient bandwidth (by providing a proxy server which compresses the data); or 3) the support of unstable connections (by providing a local service proxy, which re-synchronises with the remaining servers after connection re-establishment). The proposed algorithm is optimised to deal with the first problem.

A good measure of the quality of service for this system is the global service response time to client requests. For each application, this time must be confined within specific bounds. It includes: the time to resolve the application name to a server reference (at the location service), plus a waiting time due to client load, plus a service time dependent on the application (which depends on whether it is a single RPC or a session, on the overheads for distributed data consistency, etc.).

The adaptation algorithm aims at controlling the number of server agents deployed, but also their locations and each server domain, to keep the waiting time under control due to client load. The most important parameter will be the number of servers. However, the others will influence the relative distribution of clients, and the load induced

at the location servers. As clients are bound to servers based on the distance, some unbalancing can exist depending on the relative distribution of clients.

The adaptation algorithm proposed for overloaded servers in this paper is isolated. I.e., each server monitors its local load to detect conditions which violate the application requirements (threshold values are used), and compiles the clients' origins. Any server activity measurement might be used. However, best results are achieved if the load includes a measurement of the queued client requests waiting at the server.

If the load goes above a top threshold value, the server creates and deploys a new server. The replica location is selected amongst the most frequent sources of agents (local or not). The new replica will start running after T_{clone}, which is the time to create a clone on the remote agent system, plus the delays at the location service (dissemination of the new clone's server reference). During this period, new clients continue to bind to an already overloaded server. So, the triggering mechanism of the top threshold value is disabled for a duration dependent on T_{clone}. A simple load measurement is the length of the waiting clients' queue, where the disabled period is converted in an increment of the threshold value (proportional to the clients processed by a new server during its creation). In result, the replica creation rate is controlled by the client request's arrival rate.

If the demand is very high, a server might become crowed with bounded clients (and the waiting time exceeds largely the server response time). A redistribution of clients must then take place to speed up the client's service total response time. The server can unbind some of the clients, or can mutate itself (i.e., close the old interface and create a fresh one). This will force a new resolution phase for all waiting clients and a redistribution of the clients for the available servers.

Dynamically created servers are destroyed using a market based control technique [6]. When a server's load goes below a bottom threshold, it sends messages to the other servers within a maximum range, requesting one of them to take its place. Requested server's answer with bid messages, stating if they can expand their domain and their load. To speed up the overall system response, servers are not blocked for a "bid" time. The resolution phase (load transfer phase) is acknowledged, and the load transfer can be aborted if, for instance, the server had its load increased in result of another ending server and can not handle the new clients. After a load transfer is accepted, the location service is updated, and the originating server dies.

Compared to other approaches based on scheduler objects [3],[9] or inter-server synchronisation [6], the use of an isolated algorithm during overload conditions allows a faster response, and scales to higher levels of client requests. However, it may cause a temporary deployment of a higher number of servers, if for instance, the majority of the origin of the clients moves to another region of the network. During a transition phase, new servers are created while some of the previous ones die.

A complementary algorithm can be used during moderate overloaded conditions (triggered by a lower second high threshold value), or as a response to client maximum delay overrun reports. An example for this algorithm might be a market based

control technique that would search for a underused server, prior to creating a new one. The selected server would migrate to the needed region, after delegating its previous server domain.

The presented algorithms also apply to the location servers, with slight modifications. Location servers may create a replica and split a location domain to reduce its load, or may join with another neighbour. The redistribution procedure also applies to the location servers. After a location server mutation, only agents inside the location domain will be able to reach it. Remote clients using RPC which need this location server (even indirectly) will have to wait for interface information dissemination.

3.2 Service Implementation

The proposed system is not restricted to applications which might be implemented with parallel independent servers. It can also be applied both to systems with partial non-mobility requirements (due to a static component or a huge amount of data), or to applications with intra-server synchronisation requirements.

The system is flexible in terms of mobility requirements because even applications with non-mobile components might partially use this scalable feature. If their semantics allow, several smaller interface agents can be deployed in the system. They may implement caches of information (e.g. HTTP [1],[5]), concentrators of client requests or generic proxies (e.g. [8]), reducing the non-mobile components' load.

Depending on the nature of the service, servers might need extra specific synchronisation logic to maintain consistency of shared data between the servers. The average service time will increase with the number of servers. The algorithm will only be applicable when the processing capacity gain of a new server is higher than the average service degradation. It provides better results when servers can work autonomously or have low coherence requirements. An example of the latter kind is the CODA distributed file system [11], where the coherence is implemented on limited points on time, with little or almost no degradation of service time. For applications where a stronger coherence is needed, the improvements may also come from the deployment of interface agents. The maximum intra-synchronisation rate will limit the maximum number of deployable servers, and the maximum client load supported.

4 Dynamic Behaviour

The analysis of the dynamic behaviour of the system was performed with a simulator, developed using the Ptolemy [16] simulation system. The set of tests presented focuses on the study of the adaptation to a constant demand from clients, which originates server overload. Application servers run both the server creation and server destruction algorithms. The effect of the location service was reduced by setting a low-resolution time (compared to the application service time), and by disabling the dynamic change of the hierarchy. Nevertheless, it still runs the application name distribution algorithm, which introduces a delay between the deployment of a new server

and the stabilisation of the location service information. The effects of the variation of the latency on the communication between agents were not considered, by setting it to a constant value. This very symmetrical scenario produces highly synchronised reactions on servers, but it is clearly the worst case.

4.1 Simulation Environment

All tests were conducted with the network presented in figure 1 (132 agent systems and 19 static location servers). Results were collected at the end of each measuring interval of 0.5 units of simulation time (0.5 tics). The duration of each simulation was 30 tics.

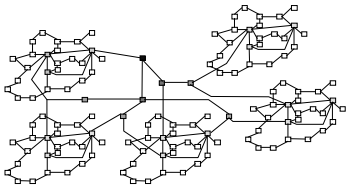

Fig. 1. Simulated meshed network

The simulation assumes an atomic interaction between clients and servers. A client is born and lives until it can make an invocation to the server. Our main results are the client's lifetimes, which are the overall application response times.

The application and location service servers are modelled by a queue defined by a service time probability function, T_S and T_L respectively. For all the experiments reported in this paper, T_L and T_S have deterministic functions with the values 0.001 and 0.1 tics respectively. The transmission time was set to 0.0001 tics, and the clone creation time to 1 tic. Servers use the number of requests on the queue as a load measurement to trigger the creation of servers, and the average utilisation time (a weighted average of measurements in 0.5 tic intervals) as a load measurement to trigger the server destruction. The top threshold value was 15 clients in queue, and the temporary increment due to clone creation was 15 clients (1.5 times T_{clone} divided by T_S). The bottom threshold value was 50% of the processing time. All the reported results were obtained with the same client redistribution procedure: clients waiting in the application server queue are unbound if their waiting time exceeds a duration of 1.5 tics. New clients were generated with a uniform distribution of the inter-server deployment time on the interval [0, 2/ClientLoad] over a group of 125 nodes. *ClientLoad* defines the average number of clients that enters into the system during a time unit.

4.2 Time Evolution

Figure 2 shows the evolution over time of some averages during each measuring interval of: *New Clients*, the number of clients which entered the system; *Unbinds*, the number of clients unbound during the interval; *Pending Clients*, the number of clients waiting on queues (of both application and location servers); *Ending Clients*, the number of clients which died during the interval; and *Processing Capacity*, the maximum number of clients that the servers can process during the interval. The second graphic shows the evolution of the average global response time per client measured on each interval, represented by TT (Total Time). The curves were measured with an average (ClientLoad) of 250 clients per tic (125 per measuring interval) and five initial servers.

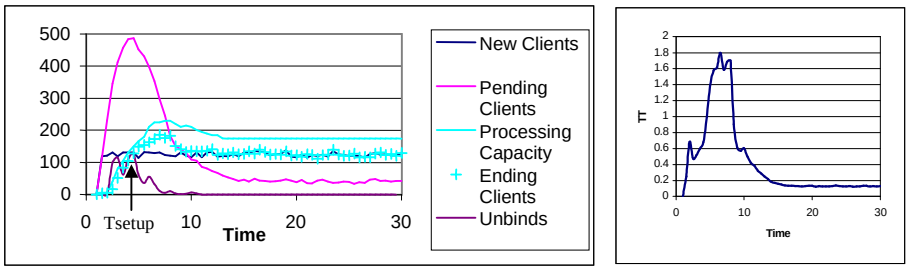

Fig. 2. Service Response – evolution on time of the number of clients and of the total delay

As soon as client requests start (at tic 1), the number of pending clients grows until a point in time where the processing power deployed is enough for the client load, T_{setup}. After that it starts to decrease. TT continues to grow just for a short while after this point (the curves are almost equal because redistribution was used, as curve *Unbinds* show). With the reduction of the number of pending clients, the number of servers decreases. However, the processing capacity is always above the new client rate due to the 50% idle time allowed for each server (the minimum load threshold). It is clear how the system gets stable with a very low and constant response time.

The time measurements used in the remaining sections were: the average value of the client lifetimes, TT_{avg}, and the time value that includes 95 percent of all client lifetimes, *TT95* (which gives a notion of how high the delay peak is). Additionally, the "Processing Capacity Ratio" (ratio between the maximum number of clients that servers can process and the number of clients entering the system) quantifies the availability of processing resources to satisfy the client demand. The variation of the client waiting time depends on the value of the processing capacity ratio (PCR) and on the distribution of clients per server. It gets higher when the PCR is below one and gets lower otherwise (assuming a completely balanced system).

4.2 Results with Weak Inter-server Synchronisation

The algorithm performance depends on a number of parameters and options, which include: the client redistribution procedure, the time to create a clone, the top threshold value, the timeout value (with partial client unbinding), the initial number of servers, and the ClientLoad. The presented results focus on the scalability with ClientLoad. A study on the effect of some of the other parameters on the server creation algorithm can be found in [4].

The next set of experiments study the system response to different client loads (ranging from 125 clients per tic to 8000 clients per tic), using two different numbers for the initial servers (initial processing capacity of 10 and 50 clients per tic). Figures 3a, 3b and 4 show the distribution of TT_{avg}, TT95 and PCR.

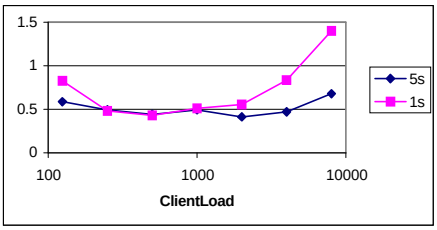

Fig. 3a. TT_{avg}

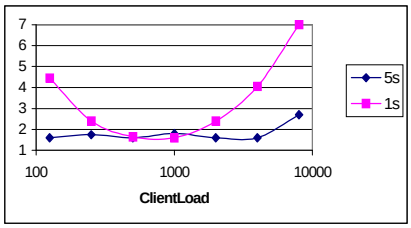

Fig. 3b. TT95

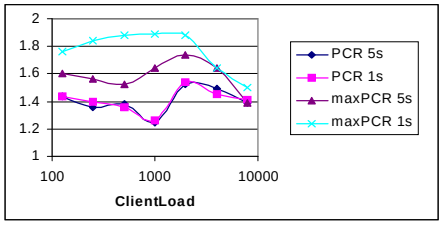

Fig. 4. PCR

The results show a minor variation of the response times (TT95 and TT_{avg}) and of PCR compared to the variation on ClientLoad (6400%), which prove the algorithm scalability. The initial number of servers has a great influence on the three parameters, except for the final number of servers deployed (PCR). Location servers (which support a maximum of 1000 requests per tic) influence negatively the system response for ClientLoad values above 1000. This effect in specially noticeable with a single starting server (but also for five starting servers for 8000 clients per tic), where the overloading of the single initial application server's location server occurs. In result, the time to resolve the application names increases, and in consequence, the server clones creation is delayed. For ClientLoad values below 1000 clients per tic, when the load is higher the system has the following characteristics: the adaptation gets done quicker (due to a flooding of servers), is less sensible to the initial number of servers, and has a lower final value of PCR (higher server usage).

The effects of slow response for low ClientLoad values and location server overloading could be compensated. The slow response could be improved by configuring the algorithm parameters (the client redistribution timeout and the top threshold) for a faster response. The use of a higher number of initial application servers would reduce the number of resolutions received at each of the location servers. Nevertheless, the dynamic change of hierarchy of the location service would create new location server replicas, and the load would be redistributed between them. After a transition phase (for local client's requests draining), the service would become available again for the entire network.

4.3 Results with Strong Inter-server Synchronisation

Most services require some state synchronisation between servers. This will introduce a limit to the maximum number of clients (load) which can be processed per tic. It is

still possible to use the algorithm on these cases with minor corrections as long as the client load is below the maximum value supported. The average service time increases when a new server is created (and decreases when one dies). When the service time increases, fewer clients are serviced per time unit. In consequence, it takes less time to reach the load top threshold value. The resulting faster clone creation might originate an explosion of server creation. The algorithm was modified to avoid this effect: the top threshold and the client timeout values are incremented when the average service time increases and decremented otherwise. It lets the system adapt more slowly to load peaks.

We tested the approach on a system with a linear degradation for each server (which models a periodic synchronisation between the servers), with the service time given by the following formula: $\text{ServiceTime} = 0.1 \times \left(1 + \alpha \times \left(\text{NumberServers} - 1\right)\right)$.

Figures 5a and 5b show the distribution of TT_{avg} and PCR to different client loads, for nine values of α ranging from 0 (no interference) to 0.07, and five initial servers.

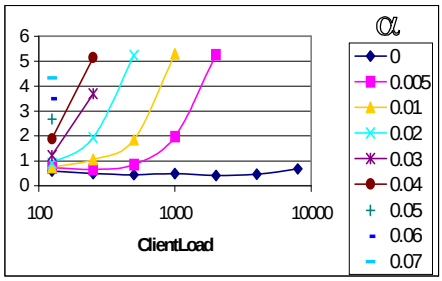

Fig. 5a. TT_{avg}

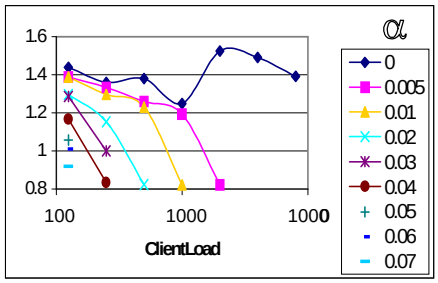

Fig. 5b. PCR at tic 30

The main effect of inter-server service degradation time is the existence of a maximum value of ClientLoad[1] for each value of α. When ClientLoad approaches the supported maximum, TT_{avg} increases and the value of PCR decreases. For some ClientLoad values below the maximum, it takes too much time to deploy the necessary servers (PCR below 1), and the system diverges. Although the maximum ClientLoad for the value 0.07 is 143, the system diverges with 125 clients per tic.

5 Related Work

The use of replicated objects indexed by a global location service to support world-wide applications is also proposed on [8], [18]. Their location service is based on a static hierarchic structure, with some scale limitations. Further, they do not handle applications with overloaded servers.

[1] MaxClientLoad = $1 / (0.1 \times \alpha)$

An algorithm to control the location of a mobile but constant set of servers is proposed on [17]. However, it only handles limited bandwidth problems.

Another approach introduces client based scalability [20] by consulting a server directory and scanning for the best available server. This approach needs some client modifications, and implements a limited architecture: the number of servers does not adapt, and each server will always interact with all clients.

6 Conclusions and Future Work

This paper presents a co-operative agent system that allows applications to scale to large networks with millions of users. The dynamic behaviour of the algorithm in face of a strong rise on client demand was studied and several conclusions were drawn based on the results. An overall conclusion is the suitability of such systems and algorithms to respond to "client peak invocations". Traditional approaches do not scale and will create severe bottlenecks if used under these conditions.

The simulation results showed that applications scale with the client load, until a limit defined by the location server capabilities. If the range of values for clone creation, service time, and for name resolution are known, then some quality of service guarantees can be assured. By the correct control on the number of replicas initially deployed and the correct setting of the algorithm parameters, an application may be ready to respond to a roughly predicted rise of the client demand. The inclusion of the dynamic change of the hierarchy of the location server will most likely reduce the dependency on the name resolution time. It is a subject under study.

The use of the mobile agent paradigm provides a sound basis to implement a dynamic service specific algorithm for server deployment. Most of the mobile agent systems available today allow the implementation of the proposed algorithm, if the location service functionality is implemented.

This paper covered atomic interactions between clients and servers. Multi-invocation interactions and session interactions can introduce other requirements to the algorithms and are being studied as well.

Acknowledgements

This research has been partially supported by the PRAXIS XXI program, under contract 2/2.1/TIT/1633/95.

References

[1] Baentsch, M., Baum, L., Molter, G., Rothkugel, S., Sturn, P.: Enhancing the web's Infrastructure: From Caching to Replication. IEEE Internet Computing, Vol. 1 No. 2, March-April (1997) 18-27

[2] Baumann, J., Hohl, F., Radouniklis, N., Rothermel, K., Straβer, M.: Communication Concepts for Mobile Agent Systems. In: Mobile Agents - Proceedings of the First International Workshop on Mobile Agents (MA'97), Germany, Springer-Verlag LNCS Vol. 1219, April (1997) 123-135
[3] BEA: TUXEDO White Paper. (1996) http://www.beasys.com/Product/tuxwp1.htm
[4] Bernardo, L., Pinto, P.: Scalable Service Deployment on Highly Populated Networks. In: Intelligent Agents to Telecommunication Applications - Proceedings Second International Workshop IATA'98, Paris, Springer-Verlag LNCS Vol. 1437, June (1998)
[5] Bestavros, A.: WWW Traffic Reduction and Load Balancing through Server-Based Caching. IEEE Concurrency, Vol 5 N 1, January-March (1997) 56-66
[6] Chavez, A., Moukas, A., Maes, P.: Challenger: A Multi-agent System for Distributed Resource Allocation. In: Proceedings of the International Conference on Autonomous Agents, Marina Del Ray, California (1997)
[7] Chia, T., Kannapan, S.: Strategically Mobile Agents. In: Mobile Agents - Proceedings of the First International Workshop on Mobile Agents (MA'97), Germany, Springer-Verlag LNCS Vol. 1219, April (1997) 149-161
[8] Condict, M., Milojicic, D., Reynolds, F., Bolinger, D.: Towards a World-Wide Civilization of Objects. In: Proceedings of the 7th ACM SIGOPS European Workshop, Ireland, September (1996)
[9] Deng, X., Liu, H.-N., Long, J., Xiao, B.: Competitive Analysis of Network Load Balancing. Journal of Parallel and Distributed Computing Vol. 40 N. 2, February (1997) 162-172
[10] IBM Aglets Workbench - Home Page. http://www.trl.ibm.co.jp/aglets/
[11] Kistler, J., Satyanarayanau, M.: Disconnected Operation in the Coda File System. ACM Transactions on Computer Systems Vol. 10(1), February (1992)
[12] ObjectSpace Voyager V1.0.1 Overview. http://wwwobjectspace.com/voyager/
[13] OMG Inc.: The Common Object Request Broker: Architecture and Specification, Rev 2.0. July (1995)
[14] OMG Inc.: Trading Service. OMG TC Document 95.10.6, October (1995)
[15] OMG Inc.: Mobile Agent Facility Specification. OMG Draft, October (1997) ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf
[16] Ptolemy project home page. http://ptolemy.eecs.berkeley.edu/
[17] Ranganathan, M., Acharya, A., Sharma, S., Saltz, J.: Network-aware Mobile Programs. Technical Report CS-TR-3659 and UMIACS TR 96-46, Department of Computer Science and UMIACS, University of Maryland, June (1996)
[18] van Steen, M., Hauck, F., Tanenbaum, A.: A Model for Worldwide Tracking of Distributed Objects. In: Proceedings TINA '96 Conference, Heidelberg, Germany, September (1996) 203-212
[19] Straβer, M., Schwehm, M.: A Performance Model for Mobile Agent Systems. In: Proceedings International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA'97, Vol. II, Las Vegas, (1997) 1132-1140
[20] Yoshikawa, C., Chun, B., Eastham, P., Vahdat, A., Anderson, T., Culler, D.: "Using Smart Clients to Build Scalable Services". In: Proceedings of the USENIX'1997, Anaheim, California, USA, January (1997)