Scalable Service Deployment on Highly Populated Networks

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. If synchronised client behaviour exists, then the limitations are even stronger. Cooperative agent systems can be a suitable technology to answer to these requirements if some aspects of the architecture are carefully designed. Particular attention should be given to the trader component and to the dynamic behaviour of the applications in response to client demand. This paper proposes a cooperative mobile agent system with a very dynamic and scalable trading service. The system is designed to allow applications to deploy servers onto the network to respond to demand making them self-configurable. Sets of simulations were performed to study the dynamic behaviour of the overall system, and identify the relevant tuning parameters.

Topics. Open, Scalable Agent Architectures for Telecommunications Applications; Middleware for Agent Communication

1 Introduction

One of the problems facing designers of client/server applications, which are deployed on large-scale networks with millions of users, is the dimensioning of the server entities. Moreover, a characteristic common to some of the applications is the possibility of a synchronous pattern on the behaviour of the clients, producing peaks of traffic on servers. Examples of such applications are easy to define and can be tele-voting, tele-shopping, real-time sports brokering, stock brokering or applications based on interactive TV interfaces. These applications will involve a large number of clients, that may produce a burst of requests after a relevant event (e.g. a team scores a point, the announcement of promotional prices, or a deadline is approaching). It is essential that the application must satisfy the service response time requirements even under these extreme conditions. Therefore, both its static and dynamic behaviours need a careful design.

An efficient design choice is the possibility of launching a variable number of servers to process client requests in parallel (assuming that the nature of the service allows server mobility). A static approach to the problem, using a fixed number of servers and conventional traders, leads to inefficient resource usage solutions: either the number of servers is insufficient or there is over-dimensioning of the servers

deployed. Most of the traditional dynamic solutions use a strictly system-based approach, instead of a per-service solution: they rely on system components to balance the requests amongst a constant pool of machines [2], [4], [6], [9], possibly deploying new servers. However, a worldwide service must not be based on a limited set of machines. Servers must be spread worldwide near the client's location to avoid bandwidth bottlenecks. The dynamics on the client's location over time will guide the server's deployment, resulting in lower client-server communication delays [20]. Such requirements can be achieved with any system providing remote object creation with state initialisation (such as factory objects or ORB implementation repositories [16]). However, the distribution of the server implementation would have to be made on all possible platforms. Mobile agent platforms provide a simpler way to deploy personalized services since all servers are obtained by cloning an initial replica.

The proposed system is based on a mobile agent system platform. Its architecture does not conflict with the standardization efforts of OMG [17], or some of the available mobile agent systems ([10], [15], etc). The main differences are the requirements of the trader (called here location service).

As it is described in this paper, our system assumes a scenario where clients look for a precise service. The system does not support service discoveries based on characteristics [13] (price, availability, etc.). Clients look for applications using a unique application name that is resolved to a server reference by the location service. Each application server sets an area of the network for its service to be known (server domain). Then, the location service splits the network dynamically into areas depending on the server population, the server domains, and the location server's own load. The location service has a new scalable algorithm to adapt to its own load (lookup requests) and to advertise the services of the servers it is responsible for. An overview of the location service is given in section 2.

Server mobile agents are autonomous on their control over server deployment. Servers use the dynamic topological information of the location service, the client load, and the overall situation of servers that belong to the application they are serving, to control the deployment of new servers and adapt precisely to the client load. The aim is to produce a highly flexible and scalable system that can support millions of interactions maintaining the desired quality of service. The proposed adaptation algorithm works as follows: each server monitors its client load and compiles the domains of the clients. When a peak on the load occurs, the server agent reacts creating clones, and deploying them based on the client origin information. A market oriented algorithm is used to reverse the process - lower the number of servers, when the client load decreases.

The study of the dynamic nature of the system covers too many aspects to be described in a single paper. In this paper we present a brief overview of the system components, the server deployment algorithm and a study of the dynamic behaviour of servers in face of a rising client demand (the adaptation to the worst case). The location service, which is itself a special dynamic service with its own load adaptation algorithm, will be covered on another paper.

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 location service.

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 the 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 servers, clients and location servers acts as a general load balancing mechanism in the system.

An application may scale and respond to peak conditions by deploying new servers, and thus increasing its total server processing capacity. The effectiveness is conditioned by the amount of time a server is inactive during the duplication process (which must be compared with the response time required by clients); and by the extra overhead to maintain consistency of the shared data due to the existence of a new server. The intra-server synchronisation is specific to each application (a simple placement of an order would not need such logic).

2.1 Location Service

The location service is one of the major players for scalability. Its requirements include: the necessity for fast updating during the creation of a server clone, the propagation of frequent updates due to server migration, and the dynamic nature of the information in the overall system (based on dynamic server domains). These requirements introduce a high overhead, which invalidates some of the current technical solutions, based on static hierarchical systems. Particularly:

- The use of cached values at remote nodes makes a fast change on the configuration information impossible [1].
- The use of full path names which define completely the search path [1], [11], [18] requires the use of a “home” server to keep all service related information (creating a bottleneck), or the backpropagation of a modification through the entire location network (making a update a costly operation).

None of these techniques must be present on a scalable and highly mobile system. First of all, the application names must be flat ([21] reached a similar conclusion). Secondly, the search path, which is now independent from the name structure, must be performed on a step-by-step basis, through a path of location servers where each one contains routing information indexed by the application name. Thirdly, this step-by-step path should be tuned by the load and characteristics of the overall system.

The routing information for the path is based on service references. References are either the full application name or incomplete information about the application just to direct the search to another location server. The objective is to keep references small and easy to update.

One important feature is how the location service scales to a large population. We use a mixture of meshed and hierarchical structure, where location servers at each hierarchical level interact with some of the others at that level and (possibly) with one above. Higher hierarchical levels always have incomplete information about the available services. Additionally, 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 size of 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). The structure of the location service hierarchy (hierarchical levels and meshes) will vary to gather the necessary number of agent systems the server wants in its domain. Higher hierarchical levels offer a broader, but less detailed vision of the services available. On the other side, clients control their search range. Due to lack of depth, or incomplete information, resolutions can fail, and a deeper search must be tried.

Figure 1 shows an example of the location service. The lowest plane, the agent system plane, has sets of agent systems forming meshes. The second and third planes show two hierarchical levels of the location service (with meshes in each one, and upward connections not designed). In this particular case, a reference to the server 's' is known completely on all agent systems at the server location's domain (darker grey area). The lighter grey area represents the scope where incomplete information is known (the reference to the location server associated with the agent system where the server is running).

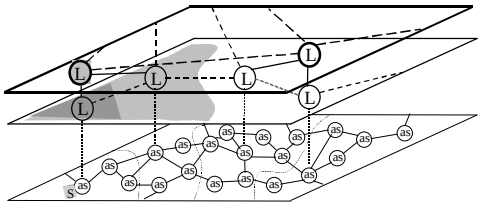

Fig. 1. Location Service Network Model

3 Server Deployment

A good measure of the quality of service for this system is the global server response time to client requests. For each application, this time must be controlled within specific bounds. It includes: the time to resolve the application name to a server reference, plus a waiting time on the server due to client load, plus a service time

dependent on the application (which depends on whether it is a single RPC or a session, the overheads for distributed data consistency, etc.).

The controllable system parameters are the number of servers deployed, their location and each of the server domains. The main parameter will be the number of servers, which defines the total available server processing capacity. At some instant, the "processing capacity ratio" (ratio between the number of servers deployed and the number of clients entering the system per time unit multiplied by the average service time) quantifies the availability of processing resources to satisfy the demand of new clients. The variation of the 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). As clients are bound to servers based on the distance and on the relative importance of the server (broader server domains get more clients), some unbalancing can exist depending on the relative distribution of clients.

3.1 Deployment Algorithm

The number of servers and their location could be configured if the service usage peaks and the origin of the requests could be anticipated. Unfortunately, for most of the applications, there is only a vague forewarn of how much the "peak load" may be, or when it will happen.

The proposed algorithm may be used with various client-server interaction methods [3]. However, it provides the best results when the interaction methods allow servers to know the precise number of clients using a specific server (server's load). This knowledge can be obtained from the client pending requests on the server's input queue (either session connections or RPC invocations). Our algorithm uses this information as well as local machine load, ignoring the number of clients that are trying to reach the server and fail. It also compiles the clients' origins and keeps a statistic indexed by the location server identifiers.

When the client load goes above a top threshold value, the server creates and deploys a new server. This action is isolated. It does not involve interaction between application servers. The new server's location server is selected amongst the most frequent sources of agents (local or not). The new server is created on an agent system picked from a list returned by the selected location server. The new server will only be completely available to run client requests 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}. When the total number of pending clients is known, a temporary increase on the top threshold value can be made to include an estimate of the number of clients that would be processed by the new server during the disabled period.

If the demand is very high, and the waiting time exceeds largely the 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). The number of pending clients (if available) or the number of consecutive load overflows are used to detect such conditions. Unbinds and mutation will force a new resolution phase for all waiting clients and a redistribution of the clients for the available servers.

A market-based control technique is used to reverse the algorithm. When the load goes below a bottom threshold, the server sends a message with a "request for bids" to all the neighbour servers (within a distance range in the location network), requesting one of them to take its place. Requested servers will answer with a bid, stating if they can expand their domain, and stating their load. The requester will wait for answers during a time interval, and selects the best bid within the minimum distance with the minimum load, afterwards. Notice that the value of the time interval is not as relevant for the system performance as in other market oriented systems (e.g. [6], [14]), where the system response time is directly related to this value. The bottom threshold is only enabled for dynamically created servers. Permanent servers (created by service provider's specification) live as long as service providers want.

The presented algorithm scales to broader networks than alternative approaches (transaction managers [4], load balancing systems [9], broker/matchmaker agents [8], or market oriented systems [6], [14]) since the adaptation to overload conditions is based on an isolated algorithm. However, it may cause the deployment of a higher number of servers since the location service balance client requests amongst servers known locally not taking into account the server's instantaneous load. The equilibrium between the client load and the number of servers deployed is achieved at limited ranges, instead of at a global level.

4 Dynamic Behaviour

The analysis of the dynamic behaviour of the system was conducted using a simulation model. The set of tests presented focused on the adaptation to a constant demand from clients. The simulation model runs all the services described so far but the interference of some algorithms was avoided. The effect of the location service was reduced by setting a very 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 stabilization of the location service information (proportional to the transmission delay between servers). The effects of the variation of the latency on the communication between agents were disabled, by setting it to a constant value. This very symmetrical scenario produces highly synchronized reactions on servers, but it is clearly the worst case.

With respect to the adaptation algorithm, we studied three alternatives for redistributing clients when a client load peak happens. The first is to mutate the server completely, to even avoid those clients that made the name resolution but are not yet

in the queue (*total unbind*). The mutation is executed after the new clone becomes operational if the queue is still overloaded. When several clones are created, deactivation is delayed until the last server starts. The second is to unbind those clients in the queue whose waiting time exceeds *Timeout* (*partial unbind*). The third is to serve all clients not doing any unbinding (*none*).

4.1 Simulation Environment

The simulator was developed using the "Discrete Event" model on the Ptolemy [19] simulation system. All tests were conducted on a network presented in figure 2, with 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. The duration of each simulation was 30 time units (*tics*).

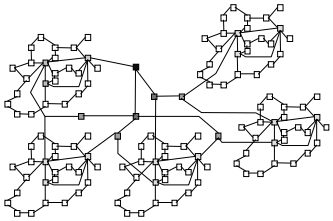

Fig. 2. Simulated meshed network

The simulation assumes an atomic interaction between the client and the server agents. A client is born and lives until it can make an invocation to the server. The total number of clients accessing a server is supposed to be known. 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 were deterministic functions with the values 0.001 and 0.1 tics respectively. The transmission time was set to 0.0001 tics. Servers use the number of requests in the queue as an indication of client load. The top threshold value is called "Maximum Client Queue Trigger Level" (*MaxCliQ*) and the disabled time after a clone is launched is 1.5 times T_{clone}.

The client creation is defined by an inter-client generation statistic, and clients are deployed with a uniform distribution to a set of 125 agent systems. The inter-client deployment statistic is defined by a uniform distribution on the interval [0, 2/ClientLoad], where *ClientLoad* defines the average number of clients that enter into the system during a time unit.

4.1.1 Results

The simulator measures the client's lifetime, the number of clients and servers, and their state. Therefore it is possible to have an evolution over time of the averages on the measuring intervals. Figure 3 shows the evolution 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 terminated during the interval; and *Processing Capacity*, the number of servers times

the service time (which measures the number of clients which can be processed on 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 ClientLoad= 250 clients per tic (125 new clients per measuring interval), using partial client unbind with T_{clone}= 1, MaxCliQ= 15 clients and Timeout= 1.5. It is also possible to calculate: the average value, TT_{avg}; the worst client life value, TT_{worst}; and the time value that includes 95 percent of all clients' lifetimes, *TT95*.

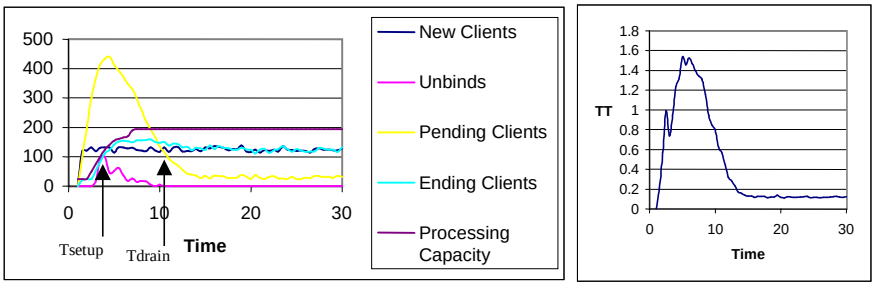

Fig. 3. Service Response – evolution on time of the number of clients and of the average total delay

As soon as client requests start (at tic 1), the number of pending clients grows and the system starts to adapt. At T_{setup} the processing power deployed is already enough for the client load. After T_{setup} the number of pending clients starts to decrease. TT continues to grow just for a short while after this point (the curves are almost equal for the experiment of figure 3 because unbind was used). T_{drain} measures the instant when the number of pending clients is below arrival rate for the measurement interval (new clients). It is clear how the system gets stable with a very low and constant response time. All the presented values are averages of at least two experiments, and can be scaled to an arbitrary system using the relation between the service time values.

4.2 Results with Weak Inter-server Synchronisation

Figure 4 present the global response time histograms for the three unbind methods.

The resulting TT_{avg} and standard deviation values for partial, total and no binds are respectively: `0.52±0.81`, `0.42±0.99` and `0.93±1.49` tics. Both unbind methods improved the system response time, with a better overall performance for the server mutation. However, the measured peak number of unbound clients is very high (`274` during a measurement interval) compared to unbound clients with partial unbind (`98`) and to the average number of new clients (`125`), resulting in PCR values of respectively `1.64`, `3.86` and `1.38` (41, 97 and 35 servers). A minimum of 25 servers was needed to satisfy the new client requests.

A particularly nasty side effect of the total unbind procedure for our very symmetrical scenario is the following: new servers receive a "peak" of both redistributed clients

and new ones, which may be higher than MaxCliQ, generating unnecessary new clones. The partial unbind method is used for the remaining experiments, because it is the one which offers the best trade-off between the deployment cost and the measured response times.

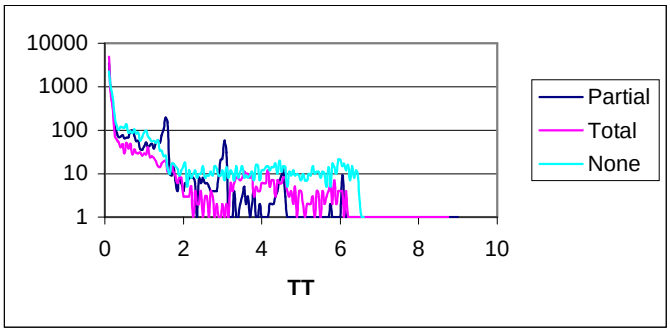

Fig. 4. Histogram[1] of response times for: partial client unbinding (with Timeout=1.5), server deactivation (total), and no unbinding, ClientLoad=250, MaxCliQ=15, five starting servers

4.2.1 Parameters Configuration

For each application, the values for service time and T_{clone} will influence the minimum achievable response time. Figure 5 shows the system performance for five values of T_{clone}, when a peak of 250 clients per tic is injected on a system with five initial servers (initial capacity of 50 clients per tic). As expected, T_{setup} depends strongly on T_{clone}, resulting in more pending clients for higher T_{clone} values. Consequently, client lifetime (TT95 and TT_{avg}) increases, and a higher number of servers is deployed (PCR). For the smallest value of T_{clone} (0.2) it is still noticeable a drain time, which exists even for T_{clone} = 0, due to buffering on the server queues (MaxCliQ is above zero).

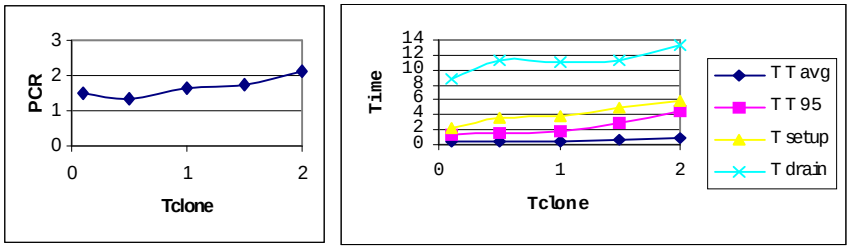

Fig. 5. T_{clone} sensibility with ClientLoad=250, MaxCliQ=15, Timeout=1.5, five initial servers

Several parameters can be set to optimise the algorithm performance. Figures 6a, 6b and 6c show the distribution of TT95, TT_{avg} and PCR for a set of 25 pairs of values for MaxCliQ and Timeout, with ClientLoad= 250 clients per time unit, T_{clone}=1 and five initial servers.

[1] `0` values were converted to `1` to allow a logarithmic representation of the number of samples

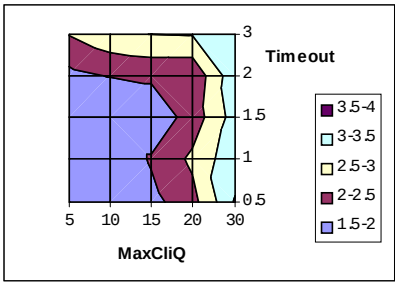

Fig. 6a. TT95

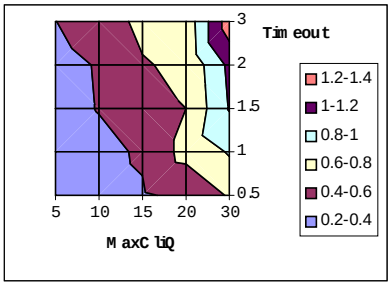

Fig. 6b. TT_{avg}

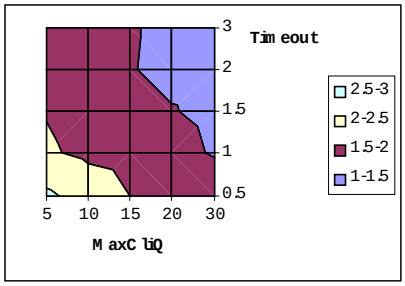

Fig. 6c. PCR

The Timeout value controls the redistribution rate between servers, and MaxCliQ controls the trigger value to launch new servers. Together, they control the system response speed to a peak of requests. The figures show a strong dependency on both parameters with an interesting trade-off: if the response time has to be very low, then PCR will rise (generating too many clones). The configuration must be defined considering the application requirements. For instance, a fast response ($TT_{avg} < 4 * T_s = 0.4$ tics) implies PCR > 2.5. The fastest measured response time (MaxCliQ=5 clients, Timeout=1 tic) had 99% of clients with a lifetime less than `2.725` tics. The best possible response time is T_{clone} plus service time (1.1), with the creation of one server for each waiting client. For Timeout values below T_{clone}, clients get unbound before a new server is deployed, which originates a high number of creations, and the measured high value for PCR.

Figures 7a, 7b and 7c show the distribution of TT95, TT_{avg} and PCR for twenty configurations of different ClientLoad and initial number of servers, with MaxCliQ=15, Timeout=1.5 and T_{clone}=1.

The results show a minor increase of the response times (TT95 and TT_{avg}) and of PCR (the ratio between number of servers deployed and the minimum necessary) compared to the increase on ClientLoad (800%), which proves the algorithm scalability. If the total number of clients bound to a server is not known then the response would be slower, and the ratio would decrease. However, it would always be low, since the number of servers deployed grows exponentially. The initial number of servers has a great influence on the three parameters, specially for lower ClientLoad values. When the load is high, the adaptation is done quicker due to a flooding of servers and the PCR gets higher (over-deployment of servers). In this case, the initial number of servers becomes irrelevant. It is interesting to note that for ClientLoad = 125 and five

initial servers the system moves smoothly and the response times are very acceptable. This proves that some load expectations from the application can produce very stable adaptations.

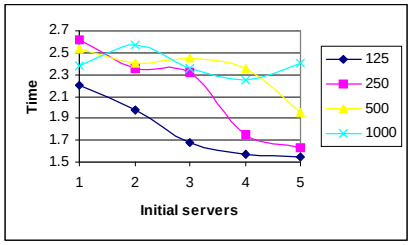

Fig. 7a. TT95

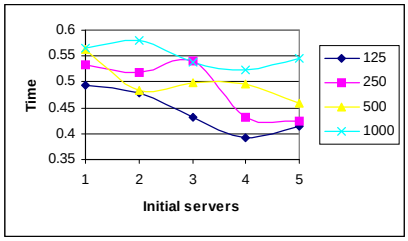

Fig. 7b. TT_{avg}

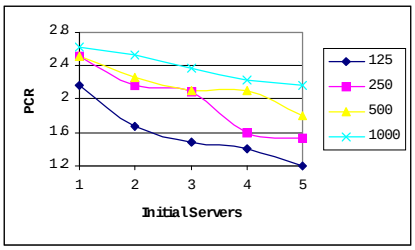

Fig. 7c. PCR

Additional experiments using random distributions (exponential, normal and uniform distributions) with the same average for T_S showed that the results were comparable to the ones measured with the deterministic value. There were slight increases on the response times (less than 20%) and on the number of servers deployed (less than 10%), inducing that the algorithm is sufficiently generic.

4.3 Server Deployment with Strong Synchronised Systems

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 time unit. 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 MaxCliQ clients waiting in the queue. The ratio between Timeout and average service time degrades until it possibly goes below 1, resulting on an explosion of server creation (a very high PCR). The algorithm was modified to avoid this effect: MaxCliQ and Timeout are incremented when the average service time increases and decremented otherwise. It lets the system adapt more slowly to peak loads.

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:

$$\text{ServiceTime} = 0.1 \times \left(1 + \alpha \times \left(\text{NumberServers} - 1\right)\right) \quad (1)$$

Table 1 shows the measurements with ClientLoad=250, T_{clone}=1, MaxCliQ=15, Timeout=1.5 (initial values), and five initial servers. The columns "Servers for T_{setup}" and "Max PCR" present the number of servers needed to sustain the incoming clients and the theoretical maximum value[2] achievable for the rate between the maximum number of clients and ClientLoad (with infinite servers). For $\alpha \geq 0.04$ the system can not support the load of 250 clients per tic (MaxPCR is below 1).

α	Servers for T_{setup}	Max PCR	T_{setup}	T_{drain}	TT_{avg}	TT95	TT_{worst}	Serv. Dep.	PCR
0	25	∞	5	11	0.50	1.9	7.86	41	1.64
0.005	29	8	5	13.25	0.63	2.1	6.85	43.5	1.5
0.01	33	4	5	22.5	0.97	2.7	9.68	49	1.49
0.02	49	2	7.5		2.40	5.9	17.78	76	1.55
0.03	97	1.33	24.5		4.12	10.4	24.21	112	1.15

Table 1. Performance with synchronised clients

Increases to the value of α lead to reduced processing capacity gains for each new server added to the system, and in result, to slower responses to client request peaks, shown by the increase of TT_{avg} and TT95. For the two higher values of α, this effect even prevents the system from draining the clients accumulated during the deployment of servers (T_{drain} is above the simulation duration of 30 tics). The difference between α=0.02 and α=0.03 is that the number of pending clients is much higher for α=0.03 and the exceeding available processing capacity (PCR) is very low (15%), resulting in higher response times (well above the simulation duration). Without the algorithm modification the system deploys ("Serv. Dep.") 284 servers with TT_{avg}=1.1 and TT95=3.2 for α=0.02, and deploys over 500 servers for α=0.03.

5. Related Work

The design of scalable systems to support worldwide applications is addressed on [7], [21], where architectures for global location services and server replica co-operation are proposed. There are some differences regarding the location service but this is not the main subject of this paper. None of them, however, handles the "client peak" invocations due to some external and uncontrolled event.

Dynamic server replication systems are proposed on [2], [5] for optimising the bandwidth usage on the access to WWW servers. The client dissemination amongst server replicas uses a "master" server, with the resulting scale limitations.

[12] proposes a low level approach based on fine-grained objects. The scalability is compromised by the lack of an "anycast group address" and by the use of front end scheduling objects.

---

[2] $MaxPCR = 1/(0.1 \times ClientLoad \times \alpha)$

Market oriented based systems [6], [14] are proposed to bind client-server interaction. However, the communication of “bid” messages between each client and all the servers presents strong scale limitations.

6. Conclusions and Future Work

This paper presents a cooperative 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 with the experiments.

An overall conclusion is the suitability of such systems and algorithms to respond to "client peak invocations". Traditional systems do not scale and will create severe bottlenecks if used under these conditions.

The simulation results showed that it is possible to guarantee a limited application response time for 95 per cent of the clients if some applications parameters are known. Namely, the maximum values for the service time, for the clone deployment time (related to the transmission time of the server agent, the bandwidth and the computational resources of agent systems), and for the name resolution time. By a correct control on the number of replicas initially deployed and the correct setting of Timeout and MaxCliQ, an application may be ready to respond to a roughly predicted rise on the client demand. Even when the demand is not predicted the system reacts well, but sometimes, it can create too many servers due to a rapid increase on the offered capacity.

The algorithm is used in conjunction with a location service, which supports the scalable trading between clients and servers. The mutual relation between both mechanisms is a challenging issue and a good direction for further study.

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

Client implementation is also a concern. Clients may communicate from their agent systems, or can also be implemented as mobile agents, migrating to agent systems nearer the server they want to use. It will be interesting to see the trade-offs between both approaches.

Acknowledgements

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

References

[1] Albitz, P., Liu, C.: DNS & BIND. O'Reilly &Associates Inc. (1996)

[2] 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

[3] 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, April (1997) 123-135

[4] BEA: TUXEDO White Paper. (1996) http://www.beasys.com/Product/tuxwp1.htm

[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] 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)

[8] Decker, K.: Matchmaking and Brokering. In: Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), May (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] ISO/IEC: Information technology - Open Distributed Processing - The Directory - Overview of concepts, models, and services. ISO/IEC DIS 9594-1, ITU-T Rec. X.500. November (1993)

[12] Kim, W., Agha, G.: Efficient Support of Location Transparency in Concurrent Object-Oriented Programming Languages. In: Proceedings of the Supercomputing'95, San Diego, December (1995)

[13] Lynch, C.: Network Information Resource Discovery: An Overview of Current Issues. IEEE Journal on Selected Areas in Communications Vol. 13 N. 8, October (1995) 1505-1522

[14] Mullen, T., Wellman, M.P.: A simple Computational Market for Network Information Services. In: Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS'95), San Francisco, June (1995) 283-289

[15] ObjectSpace Voyager V1.0.1 Overview. http://wwwobjectspace.com/voyager/

[16] OMG Inc.: The Common Object Request Broker: Architecture and Specification, Rev 2.0. July (1995)

[17] OMG Inc.: Mobile Agent Facility Specification. OMG Draft, October (1997) ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf

[18] OMG Inc.: Trading Service. OMG TC Document 95.10.6, October (1995)

[19] Ptolemy project home page. http://ptolemy.eecs.berkeley.edu/

[20] 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)

[21] van Steen, M., Hauck, F., Tanenbaum, A.: A Model for Worldwide Tracking of Distributed Objects. In: Proc. TINA '96 Conference, Heidelberg, Germany, September (1996) 203-212