Scalability of applications on active overlay networks with dynamic server deployment

Luis Bernardo

Paulo Pinto

Tele-DEE-UNL **TR-2003-01**

February 2003

Secção Telecomunicações
Departamento de Engenharia Electrotécnica
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa
Quinta da Torre, 2829-516 Caparica
Portugal

Technical reports are available at http://tele1.dee.fct.unl.pt/tech-reports.
The fies are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Abstract— **This paper presents a performance study about an algorithm to control the dynamic deployment of application servers on an active network as a mechanism to make a scalable system. Application servers measure their load and create replicas when they become overloaded. Clients can give up after a certain waiting time and return to the system for better service. This study is general enough to be applied to most of the existing systems and not only our own. The algorithm is first analyzed from an analytical point of view. This analysis highlights the importance of the various parameters and provides an indication of their values in order to configure a real system. We prove that in dynamic systems the redistribution of clients is more effective in improving performance than the extensive use of load balancing. The algorithm is finally tested on a simulator. Two experiments are presented which study the algorithm scalability and assert the analytical model.**

Index Terms—**algorithms and applications; performance evaluation, dynamic replication, grid computing**

I. INTRODUCTION

Overload situations in the Internet are already rather common and the reality will tend to get worse as the number of active users increases. Nowadays, the very small percentage of the potential users can create already annoying bottlenecks. Certain types of transactions and functionalities can use novel approaches in order to solve most of the problems. Popular web sites are generally crowded (and even more on some notorious events) and server technology [Bes97][Col98][Sch00] cannot prevent bandwidth bottlenecks on the access network. Sites that were unknown minutes before become so popular that no telecommunication infrastructure can be installed on the fly. A useful technology to tackle the problem can be the definition of active overlay networks and dynamic server deployment. The main idea is to sense the "system" and act accordingly with tools that have more or less the same time reaction as the load.

In general terms, agent technology [Mil99] can address the problem, but can these systems cope with the

huge number of users themselves? I.e. are they scalable? If the wanted objects, or the transaction processing, are heavy weight, mirror servers can be established in advance and some work has been done on managing systems for these scenarios [Ste99][COR02]. What about if we are dealing with minor entities that can suddenly become very popular? For instance a voting system attached to a TV program; the lotto betting just before the draw; availability of a parking place on the downtown parking lot; etc. Placing servers in the way traditional middleware systems do [COR02] can be very heavy and useless most of the time. If we assume that servers are very lightweight, dynamically deploying them can make sense. The system will be so mobile that an active overlay network is essential. Is such system scalable? Can it adapt to peaks of user load, both expected and unexpected? Can a system manager initially configure something to make the system more adaptable, based on the knowledge of the phenomena? What are the important characteristics of a system that enable such an approach?

This paper derives a performance analysis for such a system in order to understand the relevance of the various system parameters for the answers to these questions. It starts by presenting the scenario and a small overview of the system, both in terms of the server algorithms and adaptation model. A more extensive description can be found in [Ber98]. Simulations were performed that both validate the analytical model and showed the suitability of such a system to the presented problem. Some results are presented in the paper.

II. SYSTEM OVERVIEW

A. Context

We assume a network composed of computers and communication links between them. Each computer is a peer node where system and application objects may run. Middleware services connect the computers, and manage the network and computation resources, creating an active overlay network. A user or an active object may start one or more application servers on any region of the network, in order to deploy a new application. Thereafter, application servers will autonomously handle the deployment of new

application servers, in order to satisfy the service response time requirements.

Users access the application services using an application unique identifier (AUI), invoking a server replica. The AUI is resolved by the location service [Ste98][Cza01][Ber99], which returns a reference to the interface of one (or more) replica.

When an application becomes popular, the load is felt by both the location service and the application service. They both react by creating new replicas of the servers. Additionally, the location service can perform some load balancing when assigning server replicas. We assume no inter-synchronization between application servers. So, the performance of an application on an active overlay network depends not only on the number of application servers present on the network, but also on their locations, and on the design choices of the location service. The adaptation algorithm must respond promptly to load peaks, in order to reduce the interval when the client load exceeds the total (or local) server capacity. Additionally, the algorithm must create a number of replicas capable of handling the total load.

The location service can distribute the clients by the replicas just to minimize the usage of network resources, can perform some load balancing, or simply, distribute them uniformly on a random manner. On a very large network, with millions of users, the first option must always hold – the system must select one of the nearest replicas and avoid the use of network core bandwidth. The load balancing approach tries to keep the load uniform amongst all the replicas. However, load balancing theory results prove that for very long transmission delays the performance of a random distribution system resembles the performance of a system with periodic load update [Bil97]. Therefore, in order to produce a scalable system on a very large network, we must restrict the dissemination of load information to small regions.

B. Algorithm

In order to adapt the number and location of the application server replicas, each server monitors its local load, and possibly, the neighbor's load, to decide when to create a new replica, to migrate, or to self-destroy. There are, again, similarities with the load-balancing problem: the server decides to distribute a set

of client requests indirectly by creating a remote server replica. Of course, the problem is more complex, because the number of processors varies with time. Experiences with load balancing algorithms [Kun91][Col98] showed that load metrics based on averages of values produced slower adaptation to load, which resulted on lower performances. Load may change rapidly, and an average metrics continues to account for old values during some time. The number of needed servers can be calculated from: (a) the increase rate of the number of requests on the server queue and (b) the average request processing time ($1/\mu$). However, derivative systems can easily be very unstable. So, to avoid instability, an average of the variation rate of the number of clients on the queue during an interval with duration equal to the average clone creation time ($\tilde{t}_{clone}$), is used instead.

Several threshold values on the queue length mark the reaction moments of the servers. Reactions other than the first must take into consideration that the system is not empty. The general expression is (1), and $\beta \mu \tilde{t}_{clone}$ provides the offset, where β is a real number (usually below one).

$$Threshold(i) = MaxCliQ + (i-1)\beta \mu \tilde{t}_{clone} \qquad (1)$$

Equation (1) with β equal to one creates approximately the minimum number of server replicas adapted to the rate of new clients during an interval of duration $\tilde{t}_{clone}$. When β is bellow one, more servers are created. We will see that with such a mechanism the system will adapt to load peaks within an interval of about $2\tilde{t}_{clone}$ time units.

The new replicas will be created on a region of the network where clients are located. Servers keep a statistical record of the client's regions. Obviously, each time a replica is created in a certain region these records must be updated by $\mu \tilde{t}_{clone}$ (the number of requests processed during the interval of creation of the new replica). The algorithm proposed for the location service [Ber99] resolves an AUI following a path of location servers. The clients can have one of the two behaviors: (a) use remote procedure call (RPC) or an asynchronous version of RPC (e.g. CORBA messaging [COR02] and Web services [Cur02]) and invoke the local location server. They can get an indication of another location server and repeat the process.

When they finally invoke the service they provide the path they followed; (b) launch a mobile agent or use remote evaluation [Ghe97] and migrate to the places where the location servers are invoked. When they finally invoke the service they provide the migration path.

In possession of the path the application servers have topological information. For other location services, which resolve the AUI without any transient information to the client (e.g. [Cza01] [Ste98]), servers wishing to use this feature might have to get the information elsewhere. For instance, use client IP addresses and BGP routing tables' information [Kri00] [Guy95], to associate clients with an autonomous system. Note, however, that this topological information is a lateral issue in relation to the main focus of the paper. If it exists it can lead to a better placement of a server replica with regard to the usage of the core network.

If the replica creation time ($\tilde{t}_{clone}$) is high compared to the average request processing time ($1/\mu$), the number of requests accumulated in the queue can become too high, originating a slow overall adaptation to a peak of load – queued clients just have to wait. To reduce the adaptation time, it would be nice to redistribute the clients accumulated on servers' queues amongst all the available servers, including the new ones. This is simple to implement: clients only wait a certain time for responses (discarding late responses) (***T***); servers discard requests waiting in the queue for more than a certain time (***Timeout***). The values need not be the same, but we used the same value in the analytical model. Each time a client gives up it returns to the location service, and tries to locate a less loaded application server.

Notice that the behavior of the *Timeout* redistribution depends on the client implementation strategy and actual load conditions on the network. If remote invocation is used, returning clients start at the local location server again and can get a fresh application server placed there. If migration is used, returning clients forget the path and are contributing locally to the load, putting pressure for a fresh clone to be placed near the old one.

Redistribution, as it is, puts the system in the direction of stability but if the peak load is too severe there is only a slow decay on the application server's queue length – most of the client requests coming from

other regions of the network will be handled there. The process could be accelerated if the servers had knowledge of their load situation. Therefore, a new indicator is defined based on the average load (2). The variable $Load_i$ is the processor occupation within a reading interval. α weights history in comparison to the last value[1]. If the server is very loaded a local new replica is created.

$$\begin{aligned} Load_n &= \alpha \times Load_{n-1} + (1-\alpha) \times M_n \\ &= \alpha^n \times Load_0 + (1-\alpha) \times \sum_{k=0}^{n-1} \alpha^k M_{n-k} \end{aligned} \qquad (2)$$

The replica creation decision could also take into account the load on the neighbor application servers. Some authors [She98] have proposed cooperation algorithms for controlling replication, where servers periodically send information about their local load. If any of the neighbors is not loaded, the replication decision is delayed. Application servers must then redirect client requests to these servers, in order to preserve their local load within bounds. Unfortunately, this strategy can lead to delayed replica creation in result of obsolete load information [Bil97] and needs some overhead bandwidth to exchange information.

The destruction of application servers must be coordinated, to maintain a minimum number of servers running on the network. Surviving servers have to take into account raises of the load due to scarcer server offer. A cooperation algorithm was proposed in [Ber99], based on the average load metric (2).

III. Analytical Model

A. Simplified System Model

The purpose of the model is to study the system performance. It is related to the time it takes a client to run an application. A certain service, and ours particularly, is effective if the system performance is acceptable during the dynamic deployment of new application servers, in response to a load peak. To study the transitory period of adaptation we need to model the behavior of the entire system, composed by: a varying set of application servers, a continuous flow of clients, and the location service. Notice that traditional classical statistical models do not apply because they assume an equilibrium state, which is

[1] A similar result could have been achieved by sensing the length and behavior of the queue.

stationary and ergodic [Pap84].

The location service plays an essential role on the behavior of the system, since it distributes the clients amongst the servers, and thus, defines the servers' instant load. It is a known fact that the best possible performance is achieved using a single queue for all the servers, and a ubiquitous knowledge of the servers load. For an exponential inter-arrival client request distribution and an exponential distribution for the processing time on the servers, the system can be modeled by M/M/1 [Tah82]. Unfortunately, this scenario cannot be implemented on a very large network, due to the communication delay and the communication overhead. Instead, the proposed location service distributes evenly the load amongst the application servers at equal distances, or privileges the servers at shorter distances, independently of their local load. For restricted regions, some load information could be introduced. However, for simplicity, on the analytical model we will assume that application servers and clients are evenly distributed on the network and that client requests are uniformly distributed amongst the servers. Assuming the above distributions, the system could then be modeled by an M/M/*c* queue model. The instantaneous load of an application server ($\lambda_s(t)$) can be estimated using (3), for a total load of λ client requests per time unit, distributed by *c* applications servers. The equation also assumes that the location service processing time is small, compared to the application service processing time.

$$\lambda_s(t) \approx \lambda(t) / c(t) \qquad (3)$$

The study analyzes the evolution of the number of clients on the application server queues. If the average number of new clients entering the system is much larger than one, the discrete time system can be approximated by a continuous model. The client flow rate is defined by the average frequency of arrival of new client requests ($\lambda_N(t)$) and the processing capacity of the application servers (μ) can be modeled by the inverse of the average client request processing time on the application servers. It is assumed, by hypothesis, that the variance of both values is low compared to their average values.

Figure 1 presents the client's flows on the system. The total flow of clients entering the location service

($\lambda(t)$) are: the new ones ($\lambda_N(t)$); and the ones which were not processed on the application servers and returned to the location service ($\lambda_T(t)$), after the time limit defined above (T).

Under the above conditions, it is possible to deduct the number of clients on a server queue (4). It is also possible to deduct the total number of clients present in the system, waiting to be processed (5).

$$m_s(t) = \int (\lambda_s(t) - \mu)\, dt \qquad (4)$$

$$m(t) = \sum_s m_s(t) \qquad (5)$$

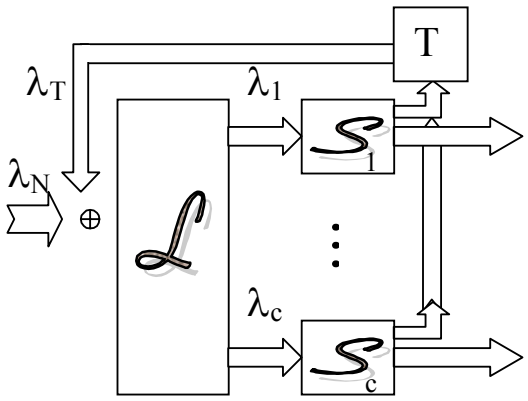

Figure 1: Client flow

This analytical model analyzes the response of a resting system, with c_0 initial application servers, which receives a peak of λ client requests after instant zero (with λ/μ greater than c_0). It analyses the influence of the application server's creation algorithm, and of the location service behavior.

B. Analysis of the application server deployment algorithm

Application servers create replicas based on the queue length and on the average load. However, during a load peak, most of the replicas will be created in result of the queue growth. The average load reacts slowly (it depends on the measurement period), detecting saturated application servers for which the queue is reducing too slowly.

Application server replicas are created whenever the number of client requests on the server's queue

$m_s\left(T_{D_i}\right)$ reaches a threshold value given by (1). In order to simplify the model the threshold parameters are assumed constants (in practical terms it speeds up replica creation a little bit). We define T_{Di} as the instant when an application server decides to create replica *i*. Replica *i* will start running t_{clone} time units afterwards, at T_{Si} (6).

$$T_{S_i} = T_{D_i} + t_{clone} \tag{6}$$

The total number of replicas created on a system depends on all the server deployment algorithm parameters, and on parameter *T*, which influences the number of clients that return to the location service. The initial set of application servers is responsible for creating most of the replicas. The creation of new replicas will only end when the server's instant load (the total load evenly divided by all the replicas) is equal or below the server's processing capacity. We define K_{max} as the number of replicas created for each initial application server (either directly or created by the ones it created). Due to the symmetry of the model, each initial server creates exactly the same number of replicas at the same instants. Figure 2 shows an example of the evolution of the number of servers in response to a load peak. It is possible to have creation decisions before or after the first group of servers has started (at instant T_{S1}). If no clients return to the location service ($T = \infty$), the number of requests on the queues of the c_0 initial servers can be calculated using (7). The evolution of the number of requests on the queue of the servers that start at instant T_{Sn} can be calculated in function of queue of the initial servers using (8). Otherwise ($T \neq \infty$), it may be necessary to add the $\lambda_T(t)$ component.

$$m_{s_0}(t) = \begin{cases} \left(\frac{\lambda}{c_0} - \mu\right)t & \text{if} \quad t \le T_{S_1} \\ \left(\frac{\lambda}{c_0} - \mu\right)T_{S_1} + \left(\frac{\lambda}{2c_0} - \mu\right)\left(t - T_{S_1}\right) & \text{if} \quad T_{S_1} < t \le T_{S_2} \\ \ldots \\ \left(\frac{\lambda}{c_0} - \mu\right)T_{S_1} + \ldots + \left(\frac{\lambda}{K_{\max}c_0} - \mu\right)\left(T_{S_{K\max}} - T_{S_{K\max-1}}\right) + \left(\frac{\lambda}{(K_{\max}+1)c_0} - \mu\right)\left(t - T_{S_{K\max}}\right) & \text{if} \quad t > T_{S_{K\max}} \end{cases} \tag{7}$$

$$m_{s_n}(t) = m_{s_0}(t) - m_{s_0}\left(T_{S_n}\right) \qquad \text{if} \quad t \ge T_{S_n} \tag{8}$$

From time zero until T_{S1} (startup of the first group of replicas) every replica creation decision will be

taken by the initial set of servers. During this period, the server queue will increase at a constant rate (λ/c_0-μ), independently of $\lambda_T(t)$. If a server discards $\lambda_T(t)/c_0$ clients per time unit due to overtime, they will be redistributed through the same set of servers, originating an equal flow of requests. The queue only grows due to the new requests, originating the creation of new replicas when its length crosses a threshold value (1). In this period, from zero to T_{S1}, the time between decisions is fixed, originating a linear relation between T_{Dn} and the number of replicas n (9). The total number of decisions to create server replicas (K_{A1}) (for each initial server) until T_{S1} can be calculated using (10), where *floor* represents the minimum integer above or equal to a number. Notice that if β is equal to or below one, the total number of servers created are capable of handling all the new requests. In figure 2, K_{A1} is equal to 3.

$$T_{D_n} = \frac{MaxCliQ + (n-1)\beta\mu\,\tilde{t}_{clone}}{\lambda/c_0 - \mu} \qquad \text{if } n \le K_{A1} \tag{9}$$

$$K_{A1} = 1 + floor\left(\frac{1}{\beta}\left(\frac{\lambda}{c_0\mu} - 1\right)\right) \tag{10}$$

After T_{S1}, the client requests will be distributed by a larger number of servers (some less loaded). If clients are not redistributed, then the queue reaches its maximum value when the processing power of servers is equal to or above the client load. Otherwise, the client redistribution will contribute to the creation of extra servers and to reduce the adaptation time. During an interval with duration T, each server of the initial group of servers will continue to send $\lambda_s(t\text{-}T)\text{-}\mu = \lambda_T(t)/c_0$ clients per time unit to the location service but will receive just $\lambda_T(t)/c_t$ requests (c_t represents the number of servers at instant t), reducing the number of clients in the queue. The new servers receive a total input rate with contributions from both λ (the new requests) and $\lambda_T(t)$ (the clients which returned to the location service). Therefore, during this interval, the new servers will create more replicas than the initial servers since they receive a higher request rate, influenced by $\lambda_T(t)$. After $T_{S1}+T$ the load of the initial servers is balanced with the load on the servers which started on T_{S1}. When a new set of servers start at T_{Sp} the client redistribution mechanism origins also

a gradual transfer of load from the existing servers to the newly created servers during an interval of duration T. The relative strength of these peaks is lower for higher values of p, because of a lower percentage variation on the processing power on the network (from $p\mu c_0$ to $(p+1)\mu c_0$). Figure 2 illustrates a typical situation when $\lambda_T(t)$ is above zero, where most of the creation decisions after T_{S1} came from the new servers.

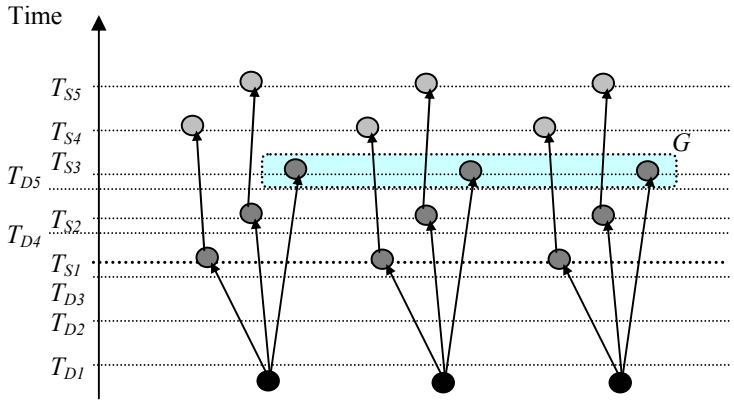

Figure 2: Server replica creation for $\beta \leq 1$

The total number of replicas created by the servers that started on T_{S_p} depends on the maximum number of clients on the servers' queues, which can be calculated using (11). The algorithm will stop deciding to create replicas when the server queues stop growing (there is already enough processing power). This happens when the total number of replicas (originated from the original servers and from their descendents) reaches $K_{setup}(\lambda_T)$ (12). For the example of figure 2, $K_{setup}(\lambda_T)=3$ (after that there are no more decisions to create replicas). K_{A1} is the initial and practical reaction of the system (not taking into account redistribution). K_{setup} is the theoretical number of servers needed to respond to the load (including the redistribution load). It can be shown that the maximum values for the queue length on the initial servers happens just before clients start going back to the location service, which depends on the value of T.

$$m_{\max}[p](\lambda_T) = \sum_{n=p}^{K_{setup}-1} \left(\frac{\lambda + \tilde{\lambda}_T[n]}{(n+1)c_0} - \mu \right) \left(T_{S_{n+1}} - T_{S_n} \right) \qquad (11)$$

$$K_{setup}(\lambda_T) = ceil\left(\frac{\lambda + \lambda_T}{c_0 \mu}\right) - 1 \qquad (12)$$

Equation (11) is impossible to calculate due to the behavior of $\tilde{\lambda}_T[n]$. It can, however be approximated in function of *T*. It can be simplified to (13) considering only the replicas creation decisions taken until T_{S_1} and for values of *T* that make $\lambda_T(t)$ constant during interval between TS_1 and TS_{KA1}. Those values are $T \approx TS_1$ and $T > TS_{KA1}$.

$$m_{\max}[p](\lambda_T) \geq \frac{\beta \mu \tilde{t}_{clone}}{\lambda / c_0 - \mu}\left(\frac{\lambda + \lambda_T}{c_0}\left(\sum_{n=p}^{K_{A1}-1}\frac{1}{n+1}\right) - \mu(K_{A1} - p)\right) \qquad (13)$$

For values of *T* such that $TS_1 < T < TS_{KA1}$, $\lambda_T(t)$ will be zero until *T*. The last server created until *T* is $n(T)$ (14). The total number of clients accumulated until *T* is (15). From *T* until TS_{KA1} further clients join the queue (16). The total number of accumulated clients is the sum of (15) plus (16), except for the initial set of servers, where the maximum length happens (17) exactly when clients start going back to the location service.

$$n(T) = 1 + floor\left(\frac{(\lambda - \mu c_0)(T - t_{clone}) - c_0 \times MaxCliQ}{\beta \mu c_0 t_{clone}}\right) \qquad (14)$$

$$m_{\max}\{highT - before\}[p] = \begin{cases} \frac{\beta \mu \tilde{t}_{clone}}{\lambda / c_0 - \mu}\left(\left(\sum_{n=p}^{n(T)-1}\left(\frac{\lambda}{c_0(n+1)}\right) - \mu\right)u(n(T) - p) + \left(\frac{\lambda}{(n(T)+1)c_0} - \mu\right)(T - T_{S_{n(T)}})\right) & if\ p \leq n(T) \\ 0 & if\ p > n(T) \end{cases} \qquad (15)$$

$$m_{\max}\{highT - after\}[p] = \begin{cases} \frac{\beta \mu \tilde{t}_{clone}}{\lambda / c_0 - \mu}\left(\sum_{n=n(T)+1}^{K_{A1}-1}\left(\frac{\lambda + \lambda_T}{c_0(n+1)} - \mu\right)\right) + \left(\frac{\lambda + \lambda_T}{(n(T)+1)c_0} - \mu\right)(T_{S_{n(T)+1}} - T) & if\ p \leq n(T) \\ \frac{\beta \mu \tilde{t}_{clone}}{\lambda / c_0 - \mu}\left(\sum_{n=p}^{K_{A1}-1}\left(\frac{\lambda + \lambda_T}{c_0(n+1)} - \mu\right)\right) & if\ p > n(T) \end{cases} \qquad (16)$$

$$m_{\max}\{higtT\}[0] = \left(\lambda / c_0 - \mu\right)t_{clone} + m_{\max}\{highT - before\}[1] \qquad (17)$$

When $T \leq TS_l\text{-}TS_{l-1}$, all clients are redistributed amongst the available servers after *T* time units, and all servers on the system have the same queue length before a new set of servers start. Therefore, the

maximum queue length is achieved exactly at the end of the interval just before a new set of server starts (18). $u[n]$ returns one for n higher or equal to zero, and zero otherwise.

$$m_{\max}\left\{T \le \frac{\beta\mu\tilde{t}_{clone}}{\lambda/c_0 - \mu}\right\}[p] = \frac{1}{(p+1)c_0}\left[(\lambda - \mu c_0)T_{S_1} + \frac{\beta\mu\tilde{t}_{clone}}{\lambda/c_0 - \mu}\left(\sum_{n=1}^{p}(\lambda - \mu(n+1)c_0)\right)u[p-1]\right] \quad (18)$$

Finally, when $T > TS_l\text{-}TS_{l-1}$ but $T < TS_l$, the system behavior involves multiple client redistribution occurring in parallel, which could be modeled by a complex model (more complex than the system analyzed in annex, for a single creation of servers). A linear approximation (19) is used instead, to calculate an approximate value for the maximum queue length.

$$m_{\max}\left\{\frac{\beta\mu\tilde{t}_{clone}}{\lambda/c_0 - \mu} < T < T_{S_1}\right\}[p] = \frac{m_{\max}\left\{T \le \frac{\beta\mu\tilde{t}_{clone}}{\lambda/c_0 - \mu}\right\}[p] - m_{\max}\{T = T_{S_1}\}[p]}{\frac{\beta\mu\tilde{t}_{clone}}{\lambda/c_0 - \mu} - T_{S_1}}(T - T_{S_1}) + m_{\max}\{T = T_{S_1}\}[p] \quad (19)$$

The total number of replicas created by each element of a p^{th} server group ($N[p]$) can be calculated resolving (20), leading to (21).

$$m_{\max}[p](\lambda_T) = MaxCliQ + (N[p]-1)\beta\mu\tilde{t}_{clone} \quad (20)$$

$$N[p] = floor\left(\frac{m_{\max}[p] - MaxCliQ}{\beta\mu\tilde{t}_{clone}}\right) + 1 \quad for \quad 0 \le p < K_{setup} - 1 \quad (21)$$

The total number of servers (c_1) created after a load peak results from the addition of all the replicas created during the adaptation, multiplied by the initial number of servers (22).

$$c_1 \ge c_0\left(1 + \sum_{p=0}^{K_{A1}-1} N[p]\right) \quad (22)$$

The proposed algorithm is flexible, allowing an extended set of configurations. Figure 3 presents the evolution of c_1 in function of *MaxCliQ*, for t_{clone} =1 tic (time units), c_0 = 1 server, λ = 625 clients/tic, μ=100 clients/tic, and for three values of T (∞, TS_1 and 0.1[2]) and two values of β (1 and 0.8). The total number of servers is influenced by *MaxCliQ* and β, but the most important parameter is T. Using the model, it is

[2] 0.1 was chosen to make T< $TS_l\text{-}TS_{l-1}$

possible to configure these three parameters in a real system to make it behave as we desire (assuming λ, t_{clone}, and μ are known). The total server deployment time is independent of the configuration – i.e., all c_1 servers will be running at instant $TS_{KA1}+t_{clone}$. However, the time clients wait before being processed on a server depends on c_1. As is demonstrated in the next section, c_1 must be large enough to handle the new clients and also the clients accumulated during the deployment of the servers, in order to have low system stabilization times.

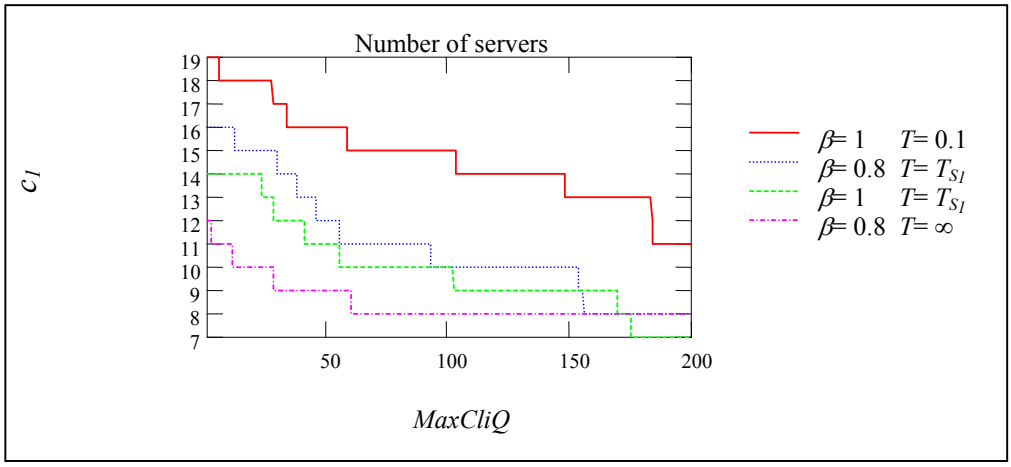

Figure 3 : Total number of servers created in response to a load peak

IV. Analysis of the System Behavior

A. Simplified model

The application server deployment algorithm originates a gradual creation of server replicas during an adaptation to a load peak. We intend to prove the stability and convergence of the resulting system, by analyzing the worst service times measured by the application clients. In order to simplify the analytical model, in this section we assume that all c_1-c_0 servers are created at $\boldsymbol{t_{run}} = TS_{K_{A1}+} t_{clone}$. This value constitutes an upper bound for the creation time of the last of the servers considered in (19). Figure 4 illustrates the approximation.

On the following sections, we will analyze the influence of the location service and parameter T on the

application performance.

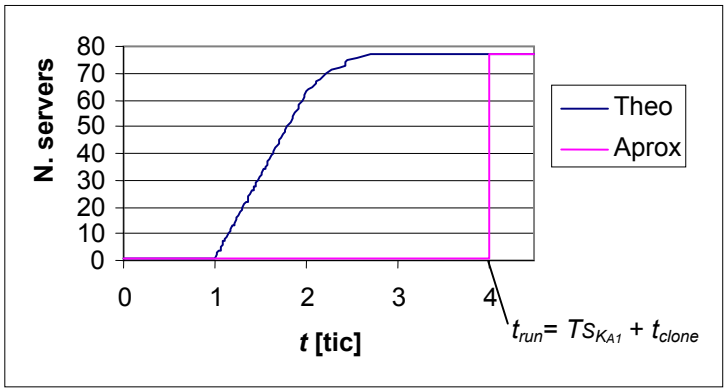

Figure 4: Approximation for the evolution on the number of servers for *MaxCliQ* = 10 clients, t_{clone}= 1 tic, c_0= 1 server, λ= 625 clients/tic, μ= 10 clients/tic, β= 1 and T= ∞ tics

B. Perfect load balancing

The theoretical best performance is achieved when a single queue is used for all the servers. This system is equivalent to an M/M/*c* queue system, which distributes the client requests to the servers whenever they become idle. The global queue evolution can be modeled by (23), where $c(t)$ models the number of servers available. Clients accumulate on the queue until t_{run}, when c_1-c_0 new servers start running. Afterwards, the number of clients decays until it reaches zero, and all accumulated clients are processed. The stabilization time (t_{stab}) can be calculated using (24).

$$m(t) = \int (\lambda(t) - c(t)\mu)\, dt = \begin{cases} \int_0^t (\lambda - c_0\,\mu)\, dt = (\lambda - c_0\,\mu)t & t \le t_{run} \\ \int_0^{t_{rune}} (\lambda - c_0\,\mu)\, dt + \int_{t_{run}}^{t} (\lambda - c_1\,\mu)\, dt = (\lambda - c_0\,\mu)t_{run} + (\lambda - c_1\,\mu)(t - t_{run}) & t_{run} < t \le t_{stab} \\ 0 & t > t_{stab} \end{cases} \quad (23)$$

$$t_{stab-optim} = \left(1 + \frac{\lambda - c_0\,\mu}{c_1\,\mu - \lambda}\right) t_{run} \quad (24)$$

However, it is not possible to implement a single queue for a large network with complete knowledge of the server loads. Therefore, $t_{stab\text{-}optim}$ will be used for comparing the performance of other implementations. Notice that t_{run} bounds the minimum achievable value, when c_1 grows infinitely (when a server is available

for each client request).

C. *Uniform Client Distribution During Initial Name Resolution*

On a large-scale system the location service will have no information about the servers load. In this section we will assume that client requests are distributed evenly by all the active servers, and that clients never return to the location service after binding to an application server ($\lambda_I(t)$ is zero).

The performance bottleneck will be on the initial set of c_0 application servers. From instant zero to instant t_{run} they will receive a continuous flow of λ clients per tic, which will stay in their queues. The system only reaches equilibrium when the c_0 servers process all these requests plus the extra requests accumulated after t_{run}. Equation (25) models the evolution of their queue lengths.

$$m_{s_i}(t) = \begin{cases} \frac{\lambda}{c_0} t - \mu t & if \quad t \le t_{run} \\ \frac{\lambda}{c_0} t_{run} + \frac{\lambda}{c_1}(t - t_{run}) - \mu t & if \quad t > t_{run} \end{cases} \qquad (25)$$

The stabilization time is calculated from the equation above, resolving $m_{s_1}(t) = 0$:

$$t_{stab-1} = \frac{\lambda\left(c_1 / c_0 - 1\right)}{c_1 \mu - \lambda} t_{run} \qquad (26)$$

You may notice that t_{stab-1} has a lower possible bound of $\lambda t_{run}/\mu c_0$, when c_1 grows infinitely.

It is possible to calculate the total delay for each client that enters the application server queue on one of the initial servers at t_i, starts being processed at instant t_o, and ends running the application at instant $t_o + 1/\mu$. With expression (27) we can calculate t_o from the entering order in the queue (m). With expression (28) we can calculate t_i for the m^{th} client.

$$t_o(m) = \frac{m}{\mu} \qquad (27)$$

$$t_i(m) = \begin{cases} \frac{c_0}{\lambda} m & if \quad m \le \frac{\lambda\, t_{run}}{c_0} \\ \frac{c_1}{\lambda} m + \left(1 - \frac{c_1}{c_0}\right) t_{run} & if \quad m > \frac{\lambda\, t_{run}}{c_0} \end{cases} \qquad (28)$$

The client delay (29) is calculated from the two expressions above, referencing both variables to the output time. Notice that (29) is only valid as long as there are clients accumulated in the queue. After $t_{stab\text{-}1}$ the average output rate of an application server is equal to the input rate, and the client delay becomes equal to the processing delay plus a possible queuing delay (it depends on the client inter-arrival statistical distribution). However, the flow model does not allow us to calculate this queuing delay.

$$delay_1(t) = t_o(\mu t) - t_i(\mu t) + {}^{1}\!/_{\mu} = \begin{cases} \frac{1}{\mu} + \left(1 - \frac{c_0\,\mu}{\lambda}\right) t & if \quad t \le \frac{\lambda\, t_{run}}{c_0\,\mu} \\ \frac{1}{\mu} + \left(1 - \frac{c_1\,\mu}{\lambda}\right) t + \left(\frac{c_1}{c_0} - 1\right) t_{run} & if \quad t_{stab-1} > t > \frac{\lambda\, t_{tun}}{c_0\,\mu} \end{cases} \qquad (29)$$

The maximum delay (30) occurs for the clients that entered in the queue just before t_{run}, and that exited on instant ${}^{\lambda t_{run}}\!/_{c_0\,\mu}$.

$$\max\{delay_1\} = \frac{1}{\mu} + \left(\frac{\lambda}{c_0\,\mu} - 1\right) t_{run} \qquad (30)$$

D. Balancing Clients During Initial Name Resolution

To assert the relative importance of load balancing and client redistribution amongst the new application servers, we studied a second scenario. Clients are assigned to the least loaded server but once in the queue, they do not go back to the location service. The big change compared to the previous scenario is only after t_{run}. All new client requests will go to the new set of c_1-c_0 servers, until load is balanced on both sets of servers (at instant t_{eq}). From then on, load is equally distributed amongst all servers.

If the c_1-c_0 new servers are not enough to handle λ client requests per tic, then the queue length on the new servers will grow from t_{run} until $m_{new}(t_{eq}) = m_{init}(t_{eq})$ (31). The evolution of the queue on the initial set of applications servers (32) will then have three sections. If they are enough, it will have only the first two

sections. Unfortunately, until t_{run} the evolution of the number of client requests on the initial servers' queue (32) is not modified. I.e. there is no redistribution and it will have impact on the delay.

$$t_{eq} = \left(\frac{c_1}{c_0} - \frac{(c_1 - c_0)\mu}{\lambda} \right) t_{run} \tag{31}$$

$$m_{s_i}(t) = \begin{cases} \left(\frac{\lambda}{c_0} - \mu \right) t & if \quad t \le t_{run} \\ \frac{\lambda}{c_0} t_{run} - \mu t & if \quad t_{run} < t \le t_{eq} \\ \frac{\lambda}{c_0} t_{run} + \frac{\lambda}{c_1} (t - t_{eq}) - \mu t & if \quad t > t_{eq} \end{cases} \tag{32}$$

The stabilization time (33) is slightly reduced compared to (26). But its minimum value (for large values of c_1) has not changed.

$$t_{stab-2} = \begin{cases} \frac{\lambda}{c_0 \mu} t_{run} & if \quad c_1 - c_0 \ge \frac{\lambda}{\mu} \\ \frac{(c_1 - c_0)\mu}{c_1 \mu - \lambda} t_{run} & if \quad c_1 - c_0 < \frac{\lambda}{\mu} \end{cases} \tag{33}$$

The same conclusions apply to the client requests delay (34). Comparing it to the delay without using load balancing (29), we notice a faster decay on the delay after the maximum value. Unfortunately, the maximum value is yet the same (30), and is related to the last client request, which entered on the queue slightly before t_{run}. If clients are not redistributed amongst all the new servers, it is not possible to improve the application performance.

$$delay_2(t) = t_o(\mu t) - t_i(\mu t) + 1/\mu = \begin{cases} \frac{1}{\mu} + \left(1 - \frac{c_0 \mu}{\lambda} \right) t & if \quad t \le \frac{\lambda \, t_{run}}{c_0 \, \mu} \\ \frac{1}{\mu} + \left(1 - \frac{c_1 \mu}{\lambda} \right) t + \frac{c_1 - c_0}{\lambda} \mu \, t_{run} & if \quad t_{stab-2} > t > \frac{\lambda \, t_{run}}{c_0 \, \mu} \end{cases} \tag{34}$$

E. Redistribution of Clients

Clients should then go back to the location service. However, they should go back gradually, in order to avoid crowding the location servers with a peak of requests. The method proposed on this paper, of

defining the time limit T has this effect: clients return to the location service at the same rate as they originally arrived at the server. It is also easy to implement.

The location service receives client requests returning from the servers at a rate (35), which is a function of the server load T time units ago on each of the overloaded servers ($u(t)$ is the Heaviside function: zero for t negative and one for t positive).

$$\lambda_T(t) = \sum_s \left(\lambda_s(t-T) - \mu\right) u\left(\lambda_s(t-T) - \mu\right) \qquad (35)$$

The system behavior depends strongly on the value of T. Let $\kappa(T)$ be the entering time of the first client that can be returned after T. From instant zero until instant $\kappa(T)$ (36) the client delay in the server queue is below T. Afterwards, some clients will go back to the location service, with probability $(1-c_0\mu/\lambda)$, T time units after entering in the queue. When T is low compared to t_{run}, clients start going back to the location service when the new servers are not running yet. In result, they will bind to the same application server, creating a continuous flow of clients that circulate through the location service at a rate $\lambda_T(t)$. So, the queue length continues to grow at the same rate $(\lambda/c_0-\mu)$ (new clients). The rate of circulating clients λ_T, is constant for each of the successive periods of T (37), but raises in each period: during the interval $\kappa+T$ to $\kappa+2T$, a certain number of clients go back to the location service; in the following interval ($\kappa+2T$ to $\kappa+3T$) a similar number returns once and some of the old that return twice. Equation (37) is valid of $t< t_{run}$.

$$\kappa(T) = {c_0\mu T}\Big/{\left(\lambda - c_0\mu\right)} \qquad (36)$$

$$\lambda_T(t) = floor\left(\left(t - T - \kappa(T)\right)\Big/T\right)\left(\lambda - c_0\mu\right) u\left(t - \kappa(T)\right) \qquad (37)$$

Notice that the extreme values of T allow us to change the behavior of the system, from a single queue system (for $T = 0$) to a system where clients are not redistributed (for T above the stabilization time). The best tradeoff is achieved when clients start going back to the location service as soon as the new replicas are created. The corresponding value of T is $T_{optimal}$ (38). For large load peaks, $T_{optimal}$ is approximately equal to t_{run}.

$$T_{optimal} = \left(1 - {c_0\mu}/{\lambda}\right) t_{run} \tag{38}$$

The system behavior considering T = $T_{optimal}$ can be described by a sequence of intervals of duration T starting after t_{run}, where $\lambda_T(t)$ is constant in each interval. This system may be modeled by a discrete system apart from the first interval of duration t_{run}. Therefore, in order to obtain a linear discrete model, the system can be simplified to an equivalent one with an initial interval of duration T. Parameter $\lambda_{Ts}[1]$ is defined as the average rate of returning clients (normalized for an equivalent interval of duration T) from each server during the second interval, and is calculated using (39). The evolution of the queue length on the initial servers can be calculated using (40). The deduction of (40) is presented in the appendix.

$$\lambda_{Ts}[1] = \left(\frac{\lambda}{c_0} - \mu\right)\frac{t_{run}}{T} = \frac{\lambda}{c_0} \tag{39}$$

$$m_{s_i}(t) = \begin{cases} \lambda_{Ts}[1]\dfrac{T}{t_{run}}t & \text{if} \quad t \le t_{run} \\ \lambda_{Ts}[1]T + \left(\dfrac{\lambda}{c_1} - \mu + \left(\dfrac{c_0}{c_1} - 1\right)\lambda_{Ts}[1]\right)(t - t_{run}) & \text{if} \quad t_{run} < t \le t_{run} + T \\ \left(\dfrac{\lambda}{c_1} - \mu + \dfrac{c_0}{c_1}\lambda_{Ts}[1]\right)T + \left(\dfrac{\lambda}{c_1} - \mu\right)(t - t_{run} - T) & \text{if} \quad t \ge t_{run} + T \text{ and } c_1 < \dfrac{\lambda + c_0\lambda_{Ts}[1]}{\mu} \end{cases} \tag{40}$$

The system behavior depends on the total number of servers available, c_1, after t_{run}. If c_1 is high enough to ensure that the client request rate is below the processing capacity (μ) (41), then a maximum bound time for processing all client requests is t_{run}+T tics. If condition (41) is not valid, then the clients can possibly return several times to the location service.

$$\lambda_s(t) = \frac{\lambda_N(t) + \lambda_T(t)}{c_1} = \frac{\lambda + c_0\lambda_{Ts}[1]}{c_1} \le \mu \quad \Rightarrow \quad c_1 \ge \frac{\lambda + c_0\lambda_{Ts}[1]}{\mu} \tag{41}$$

For c_1 values above the threshold (41) (high enough capacity), the stabilization time depends on the evolution of the queue length on the initial servers. Otherwise, all the queue lengths become equal after the second interval. From (40) we can easily calculate the stabilization time for $T = T_{optimal}$ (42).

$$t_{stab-3}\left(T = T_{optimal}\right) = \begin{cases} t_{stab-1} & if \quad \max\{delay_1\} < T \\ t_{run} + \dfrac{c_1\lambda_{Ts}[1]}{c_1\mu - \lambda + (c_1 - c_0)\lambda_{Ts}[1]}T & if \quad c_1 \geq \dfrac{\lambda + c_0\lambda_{Ts}[1]}{\mu} \\ t_{run} + T + \dfrac{\lambda - c_1\mu + c_0\lambda_{Ts}[1]}{c_1\mu - \lambda}T & if \quad c_1 < \dfrac{\lambda + c_0\lambda_{Ts}[1]}{\mu} \end{cases} \qquad (42)$$

When the number of servers is very high ($c_1 \to \infty$), t_{stab-3} will have a minimum theoretical value of (43). It measures the time to redistribute the clients accumulated and not processed until t_{run}. The redistribution of clients introduces a maximum bound on the stabilization time, which did not exist previously.

$$\min\left\{t_{stab-3}\left(T = T_{optimal}\right)\right\} = t_{run} + \lambda_{Ts}[1]T \Big/ \left(\mu + \lambda_{Ts}[1]\right) = \left(2 - 2c_0\,\mu \Big/ \left(\lambda + 2c_0\,\mu\right)\right)t_{run} \qquad (43)$$

The model presented above can be extrapolated for values of T different from t_{run}. The system only exhibits a step variation after t_{run}, with a succession of intervals of duration T, when t_{run}-κ is a multiple of T. On this case, the value of $\lambda_T(t)$ grows according to equation (37) until t_{run}. On this scenario the difference equation deducted in the appendix is still valid, requiring only a new value for $\lambda_{Ts}[1]$ (44), which by definition is equal to the redistribution rate at instant t_{run}. When the quotient is not an ordinal number, the $\lambda_T(t)$ variation intervals will not be synchronized with the starting of the new replicas. Nevertheless, we also use the same approximation for this case.

$$\lambda_{Ts}[1]\left(T < T_{optimal}\right) \approx \lambda_{Ts}[1]\left(T_{optimal}\right)\frac{\left(t_{clone} - \kappa(T)\right)}{T} \qquad (44)$$

The adaptation of the previous model for values of T greater than t_{run} is more complex, because $\lambda_T(t)$ is nonzero only after t_{run} and the queue has to grow larger in order to start redistribution. If T is above $max\{delay_1\}$(30), then the system behaves as if clients were not redistributed. Otherwise, it will have an intermediate behavior between the two models. Clients start going back to the location service at instant t_{offset} (45) (remember that when $T = T_{optimal}$ (38), $t_{offset} = t_{run}$). We approximate this system with a modified version of the previous model, where t_{offset} defines the border for the first interval. In this first interval, servers receive an average rate of λ/c_0 clients/tic from zero until t_{run} and λ/c_1 from t_{run} until t_{offset}. The

remaining intervals have a constant length of T tics. Although $\lambda_T(t)$ is not constant during each interval, in the approximated model we consider it equal to the average value. $\lambda_{Ts}[1]$ is the average rate of $\lambda_T(t)$ during the second interval (46).

$$t_{offset} = T + \kappa(T) = \frac{\lambda}{\lambda - c_0 \mu} T \qquad (45)$$

$$\lambda_{Ts}[1]\left(T > T_{optimal}\right) \approx \frac{\lambda t_{run}}{T}\left(\frac{1}{c_0} - \frac{1}{c_1}\right) + \frac{t_{offset}}{T}\left(\frac{\lambda}{c_1} - \mu\right) \qquad (46)$$

The stabilization time can be calculated using (47), an extension of (42) using the above approximation. It simply takes into account the real duration of the first interval.

$$t_{stab-3}\left(T > t_{run}\right) \approx t_{stab-3} - t_{run} + t_{offset} \qquad (47)$$

It is impossible to calculate the exact client delay on this scenario because clients may return several times to the location service before they actually run the application. If (41) is verified and $T = T_{optimal}$ (38), then clients return one time at the maximum to the location service (with a probability of $c_0\mu/\lambda$), and they may re-enter on an already loaded server (with probability c_0/c_1) or on one of the new servers (with probability $1-c_0/c_1$). If equation (41) is not verified, then clients may timeout successive times (with probability $1-c_0\mu/\lambda$ for each time). On this case the stabilization time plus the processing time ($1/\mu$) can be used as a higher bound for the maximum client delay.

F. Distribution algorithm performance comparison

Figure 5 shows the evolution of the stabilization time for the algorithm with client redistribution in function of the total number of servers (c_1), for c_0=1 server, λ=625 clients/tic, μ=100 clients/tic, t_{run}=1 tic and $T = T_{optimal}$= 0.84 tic. It also shows the stabilization time for the two algorithms without redistribution, and the optimal stabilization time. The stabilization time with client redistribution (t_{stab-3}) has a much better performance than the stabilization time with balanced initial name resolution (t_{stab-2}), and does not need network resources for load information. The stabilization time with client redistribution follows closely the

optimal stabilization time ($t_{stab\text{-}optim}$) until 13 servers. The minimum distance occurs at 12.3 servers, the threshold value defined by (41), which defines the optimal configuration value. For values of c_1 above (41) the $t_{stab\text{-}3}$ stays in the proximity of $T+t_{run}$. The stabilization time with balanced initial distribution only follows closely the optimal stabilization time for very high loads (near 7), where the stabilization time is enough to balance the load on the new and on the initial servers. For higher values of c_1, $t_{stab\text{-}2}$ stabilization time is strongly influenced by the load on the c_0 initial servers, converging to the same limit as $t_{stab\text{-}1}$, six times above the optimal stabilization time.

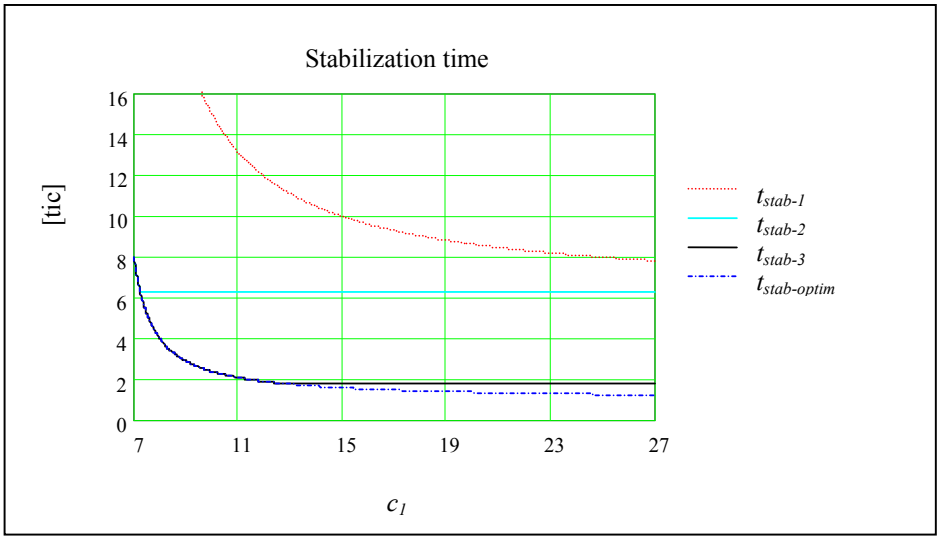

Figure 5: Stabilization time in function of c_1 for c_0=1, λ=625, μ=100, T=0.84 and t_{run}=1

Figure 6 shows the evolution of the stabilization time with client redistribution in function of T, for c_0=1 server, c_1= 13 servers, λ=625 clients/tic, μ=100 clients/tic and t_{run}= 1 tic. It shows the strong influence of T on the stabilization time. It follows approximately the optimal stabilization time when T is below $T_{optimal}$ (38). For higher values, $t_{stab\text{-}3}$ increases because client redistribution is delayed, reaching the value of $t_{stab\text{-}1}$ (uniform distribution without redistribution) for T values above 5.26 (T is above the maximum client delay). The figure also shows the model approximation error: the expected behavior would be a smooth line between $T_{optimal}$ and a point somewhere on the $t_{stab\text{-}1}$ line. The error increases with the value of T, and produces a discontinuity when T reaches the maximum delay. $T_{optimal}$ defines the best configuration because

lower values of T produce more overhead at the location service without better performance.

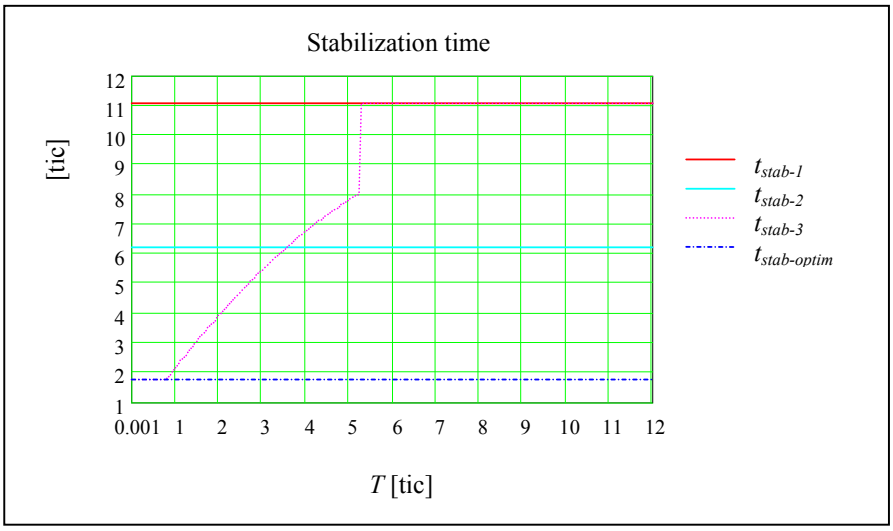

Figure 6: Stabilization time in function of T for c_0=1, c_1=13, λ=625, μ=100 and t_{run}=1

Figure 7 shows the evolution of the stabilization time for the four algorithms in function of the initial number of servers (c_0), for c_1=13 server, λ=625 clients/tic, μ=100 clients/tic, T= 0.84 tics and t_{run}= 1 tic (the optimal configuration for one initial servers identified above). Clients are only redistributed for values of c_0 below 4, otherwise the client delay does not reach T tics. The figure shows that client redistribution (t_{stab-3}) is an effective way to control the stabilization time, which is particularly effective for load peaks near the configured scenario. Although the load peak on each server is more severe for a lower number of initial servers, the better stabilization time is due to an earlier redistribution of clients (t_{offset} decreases when c_0 increases(45)). The balanced distribution of clients (t_{stab-2}) shows very good stabilization times when c_0 grows. However, it only indicates that for this amount of load the initial servers are able to handle the client before T. When they are not, c_0<3, the load balancing mechanism alone is not enough to reduce the stabilization times.

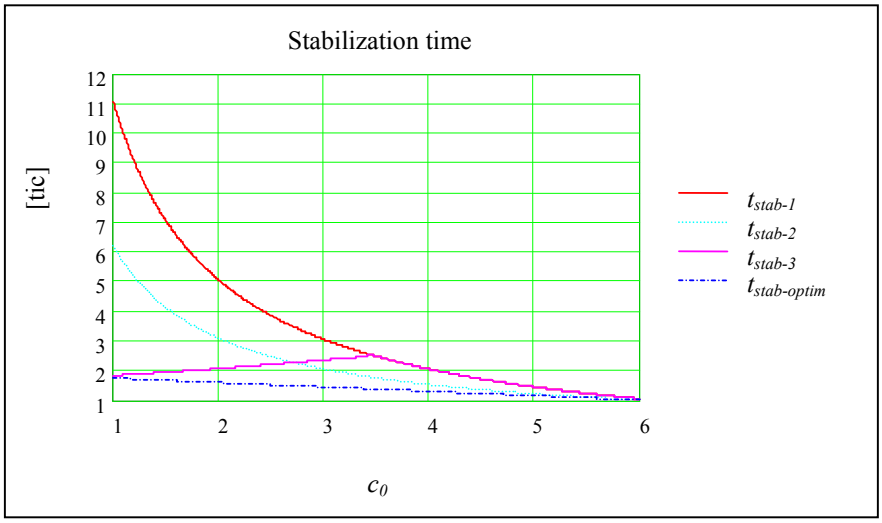

Figure 7: Stabilization time in function of c_0 for c_1=13, λ=625, μ=100, T=0.84, t_{run}=1

V. Simulation Results

A. Simulated Network

The analytical model presented on the previous sections uses several simplifying hypotheses, in order to lower the calculus complexity. The more unfeasible hypotheses on a real system are the exact load distribution through all the server replicas and the instantaneous location service. In order to study the system performance on a real system we developed a system simulator using Ptolemy simulation system [Pto97].

The simulator implements all the application deployment algorithms and a dynamic location service [Ber99], which is also adaptive to load peaks. The simulation results presented in this paper did not use this dynamic feature of the location service to avoid masking the application server own features. A static hierarchical location service, with three hierarchical layers was used instead. Servers register their interfaces on the first level location servers, which propagate registration resumes upwards creating information links on the upper layers. Searches are conducted hierarchically. Clients look up for AUIs searching location servers from the first hierarchical level upwards until they get an information link, and then follow the link downwards to the location server with a reference to the application server. No caches

are used on the location service, in order to reduce the effective application server clone creation time.

All simulations were conducted on a meshed network with 625 active nodes where applications and location servers run. Each node has an average of 3 connections to its neighbors, and the network has a maximum distance of 24 node hops. The location service used has 75 location servers at the first hierarchical level, five at the second and one at the top. Each location server can process 10,000 lookups per tic. The simulations presented in this paper evaluate the behavior of the applications when, starting at instant one (tic), a total load of 625 application clients per tic (uniformly distributed on the network) try to run the application with a single starting application server. The simulation time lasted 100 tics. During the simulation, the application servers adapt to the load accordingly to the algorithm presented in section II. Application servers measure the processor utilization time during intervals of 0.5 tic and test the average load after each measurement interval (using 50% for α). They react when the average load is above 99% or below 30%. They also measure the client request queue and react when it goes above the threshold, with β=1. Application server clone creation time (t_{clone}) is equal to one tic.

B. Results

In order to test scalability, we analyzed the behavior of the system with three different values for application servers request processing time ($1/\mu$ of 0.01, 0.1 and 0.2 tics, corresponding to S01, S1 and S2). *MaxCliQ* was set to a low value in order to get a very fast response and *T* was set to one tic. Figure 8 shows three values measured during the experiment: ($\max\{delay\}$) the maximum time experienced by the slowest client to run the application (measured from the instant the client is created to the instant when it stops running); the stabilization time; and Processing Capacity Ratio (PCR). The PCR defines the ratio between the total processing capacity available in the system ($\mu \times$ number of servers) and the client request rate (λ). Since the application and location service queues never reach zero, the stabilization time is measured when the total number of pending requests is below the average arrival rate of new request during a load measurement interval (312 requests). Figure 8 also shows the corresponding analytical values calculated

using the model of section III, respectively T01, T1 and T2.

Figure 8 shows that if no bandwidth limitations are present, the system scales well with the service processing time. The stabilization time and the maximum delay do not increase with $1/\mu$. Notice that the stabilization time improves significantly from S01 to S1 with the increase of the load in contradiction with the analytical model (T1 is only slightly better than T01). What happened was that S01 originates a smaller number of server replicas (an average of 15), whereas S1 and S2 originate a larger number of replicas (respectively an average of 141 and 255), producing some unbalancing in the server distribution on the network. This unbalancing influences the client load distribution (it minimizes the distance), and the number of application servers created. The use of a small value for *MaxCliQ* allowed us to have the maximum client delays below 7 tics (5 tics for S1 and S2), but originated a higher value for the PCR compared to the analytical model – the probability of the queue length being above two is high (especially in the presence of unbalanced load), originating the creation of additional replicas.

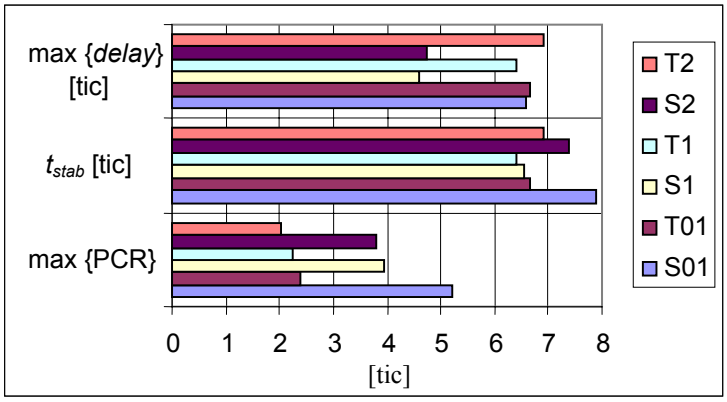

Figure 8: Simulation results (*S*) and theoretical values (*T*) for λ=625, c_0=1, t_{clone}=1, *MaxCliQ*=2, β=1, *T*=1, and for μ=100 (*01*), μ=10 (*1*) and μ=5 (*2*)

We also analyzed the influence of the clone creation time in the performance of the system, with values of t_{clone} ranging from 0.01 to 10 tics. *MaxCliQ* was set to ten and *T* was set to 1.5 tics. Figure 9 shows the analytical (lines) and measured average (bars) values for: (t_{stab}) the stabilization time; (max{*serv*}) the maximum number of servers; (t_{setup}) the time to deploy the minimum number of servers needed to handle the load ($T_{S_{KA1}}$). For the measured values, the figure shows the average value, and the minimum and

maximum values measured for each configuration. It shows a strong influence of t_{clone} on the measured values, which was already predicted in the analytical model. t_{clone} is the main factor that defines the best possible performance, which can only be compensated by a higher initial number of application servers (c_0). The analytical setup time values follow closely the measured values, because the clone creation decision is a local decision on the initial replica. However, the load unbalancing and the location service processing limitations influence both the stabilization time and the number of servers. The theoretical symmetrical model fails to capture the dynamics of temporary imbalance, due to the concentration of requests on "near" servers. The proposed algorithm handles the imbalance creating new servers in response to a high average load, or to a peak of requests above *MaxCliQ* (on the new servers). This results on a higher peak number of servers and on a slightly larger stabilization time. Notice, however, that the theoretical value for t_{stab} fall within or near the lower bound of the measured values.

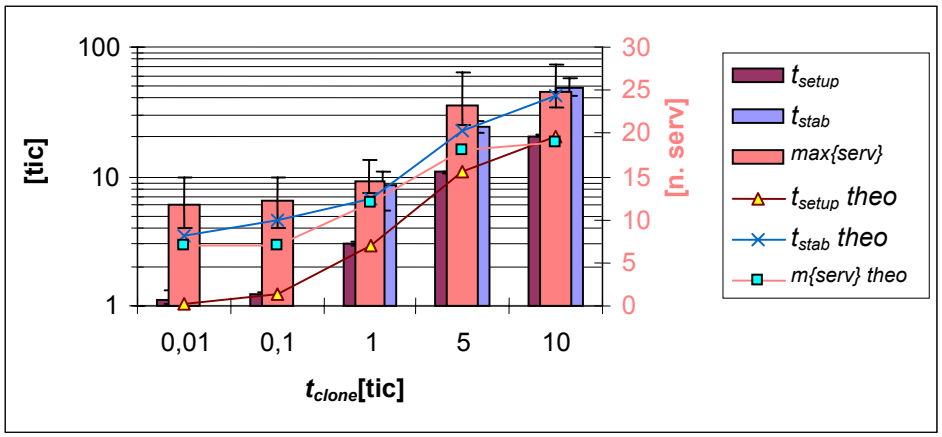

Figure 9: Simulation results and theoretical values (*theo*) for λ=625, c_0=1, μ=100, *MaxCliQ*=10, β=1, T=1.5, in function of t_{clone}

VI. Conclusion

This paper studied the performance of an application that receives a peak of client requests, and self-adapts to the load creating replicas. It shows that when applications composed by independent servers run the algorithm proposed on this paper on a network with no bandwidth problems, the application adaptation

is bounded in time and the clients delay is also bounded. It also shows the importance of the middleware component responsible for assigning clients to servers – the location service.

Several papers enforce the importance of the balancing of the load on the performance of a system [She98][Bil97][Col98]. On this paper we showed that the redistribution of the existing clients is more important than the balancing of the load when the number of servers varies. It is a cheap mechanism that only uses local interaction. The middleware role consists simply on distributing the load by minimizing the distance, thus reducing the usage of network core links and core routers. With a suitable configuration its performance is similar to a system with a single processor with the same total processing capacity. We showed that a load balancer with ubiquitous load knowledge performs worse than a system with redistribution. But, anyway, the former is impossible to implement (due to the aging of load information).

The analytical model also helps to define an initial configuration for the algorithm parameters. Simulation results showed that even with some unbalancing, the application scales with the client load, and is capable of assuring predictable bounded client delays. The most important parameter is the clone creation time. It includes not only running the application server, but also making its interface reference known in the network. If caching is used [Alb01][Ste98], the new application servers will be known only when the previous cached values become invalid, making things worse.

Section II.B presented the main ideas to support the proposed system. They can be implemented with mobile agent platforms, grid computing system, distributed component systems with mobility support, etc. However, the proposed system introduces new requirements to the middleware, which are not supported by the existing systems. Namely, avoiding caching.

APPENDIX

This appendix makes a theoretical analysis of the system behavior on the conditions described in section IV.E (when clients return to the location service after waiting T tics on a server queue). At instant zero a constant flow of λ client requests per tic start arriving at a resting system, and at instant t_{run} c_1-c_0 new

servers start running, with $c_1\mu > \lambda > c_0\mu$. When $T = T_{optimal}$, the system behavior evolves in a sequence of intervals of duration T (except the first one), which can be modeled by a discrete model.

The rate of clients returning to the location service from one server is given by the input rate on the previous interval minus its processing capacity rate ($\lambda_{Ts}[n] = x[n-1] - \mu$). The total rate of clients returning to the location service will be $\lambda_{Ts}[n]$ multiplied by the number of active servers. We can write the input rate on a single server at interval n in function of the value at interval $n-1$ (48), because the new clients request rate is known (λ). $x[0]$ includes a scaling factor to compensate for the fact that the first interval until t_{run} is slightly wider than the others. Notice that $x[n]$ decreases for values of n greater than one, until the interval where the system stabilizes (the Heaviside function is zero when $x[n-1]$ is below μ).

$$x[n] = \begin{cases} \lambda/c_0 \times t_{run}/T & if\ n = 0 \\ \lambda/c_1 + c_0/c_1 \left(x[0] - \mu\right) & if\ n = 1 \\ \lambda/c_1 + \left(x[n-1] - \mu\right) \times u\left(x[n-1] - \mu\right) & if\ n > 1 \end{cases} \quad (48)$$

Equation (48) can be resolved applying unilateral Z transform to the third element of the equation, modified by a variable shift ($p = n-2$). The resulting Z transform is (49). By inverting the Z transform and the variable shift we get the solution for $x[n]$ (50).

$$X(Z) = \frac{\lambda/c_1 - \mu}{\left(1 - Z^{-1}\right)^2} + \frac{x[-1]}{1 - Z^{-1}} \quad (49)$$

$$x[n] = x[1]u[n-1] + \left(\lambda/c_1 - \mu\right)(n-1)u[n-2] \quad (50)$$

The variation rate on the server's queue length is related to the difference between the input rate ($x[n]$) and output rate of client requests (μ and $\lambda_{Ts}[n]$) on each server queue. Equation (51) presents the variation rate ($v[n]$) on the interval $[nT\ ,\ (n+1)T]$, where n_{stab} defines the number of the interval where the system becomes stable. It is calculated using $n_{stab} = 1 + ceil\left(\left(x[1] - \mu\right)/\left(\mu - \lambda/c_1\right)\right)$, where *ceil* returns the smallest

integer above a value.

$$v[n] = \begin{cases} \lambda_T[1] & if\ n = 0 \\ \lambda/c_1 - \mu + \left(c_0/c_1 - 1\right)\lambda_T[1] & if\ n = 1 \\ \lambda/c_1 - \mu & if\ n_{stab} \geq n > 1 \\ 0 & if\ n > n_{stab} \end{cases} \quad (51)$$

The evolution on the number of requests on a queue can be obtained by integrating $v[n]$ over time (52), except for the first interval, which must be rescaled. A simplified version of this equation is presented on (40).

$$m_{s_i}(t) = \begin{cases} \left(\sum_{i=0}^{floor((t-t_{run})/T)} v[i] \times T\right) + v[floor((t-t_{run})/T)+1]\left(t - T \times floor((t-t_{run})/T) - t_{run}\right) & t > t_{run} \\ v[0]\frac{T}{t_{run}} t & t \leq t_{run} \end{cases} \quad (52)$$

REFERENCES

[Alb01] Albitz, P., Liu, C., "DNS & BIND 4th Ed."; O'Reilly &Associates Inc, Apr. 2001.

[Bes97] Bestavros, A., "WWW Traffic Reduction and Load Balancing through Server-Based Caching", IEEE Concurrency, Vol. 5 No. 1, Jan.-Mar. 1997, pp. 56-66.

[Ber98] Bernardo, L., Pinto, P., "Scalable Service Deployment using Mobile Agents", Proc. of the 2nd International Workshop on Mobile Agents (MA'98), Springer LNCS Vol. 1477, Sep. 1998, pp. 261-272.

[Ber99] Bernardo, L., Pinto, P., "A Scalable Location Service Supporting Overload Situations", Proc. Workshop Artificial Intelligence for Distributed Information Networking (AiDIN'99), AAAI Technical Report WS-99-03, Orlando, USA, Jul. 1999, pp. 51-56.

[Bil97] Billard, E. A., Pasquale, J. C., "Load balancing to adjust for proximity in some network topologies", Parallel Computing, Vol. 22 No. 14, Mar. 1997, pp. 2007-2023.

[Col98] Colajanni, M., Dias, D., e Yu, P. S., "Analysis of Task Assignement Policies in Scalable Distributed Web-Server Systems", IEEE Trans. Parallel and Distributed Systems Vol. 9 No. 6, Jun. 1998, pp. 585-600.

[COR02] OMG, "CORBA 3.0.2 specification", formal doc. 2002-12-06.

[Cur02] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S., "Unraveling the Web Services Web: An Introduction to SOAP, WDSL, and UDDI", IEEE Internet Computing Vol. 6 No. 2, Mar.-Apr. 2002, pp. 86-93.

[Cza01] Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman., C., "Grid Information Services for Distributed Resource Sharing", Proc. 10º IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, Aug. 2001.

[Ghe97] Ghezzi, C., Vigna, G., "Mobile Code Paradigms and Technologies: A Case Study", Proc. 1st International Workshop on Mobile Agents (MA'97), Springer-Verlag LNCS Vol. 1219, Berlin, Germany, April 1997, pp. 123-135.

[Guy95] Guyton, J. D., Schwartz, M., "Locating Nearby Copies of Replicated Internet Servers"; Proc. Conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM'95), ACM Press, Cambridge, USA, Aug. 1995, pp. 288-298.

[Kri00] Krishnamurthy, B., Wang, J., "On Network-Aware Clustering of Web Clients", Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'00), ACM Press, Stocholm, Aug. 2000, pp. 97-110.

[Kun91] Kuntz, T., "The influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme", IEEE Trans. Software Eng. Vol. 17 No. 7, Jul. 1991, pp. 725-730.

[Mil99] Milojicic, D., Douglis, F., Wheeler, R., "Mobility - Processes, Computers and Agents ", Addison-Wesley Longman Inc., 1999.

[Pap84] Papoulis, A., "Probability, Random Variables, and Stochastic Processes 2nd Edition", McGraw-Hill, Inc., 1984.

[Pto97] Ptolemy project home page, Dep. Elect. Eng. Univ. Berkeley, Ptolemy Classic (version 0.7). Available on *http://ptolemy.eecs.berkeley.edu/*.

[Sch00] Schroeder, T., Goddard, S., Ramamurthy, B., "Scalable Web Server Clustering Technologies", IEEE Network Vol. 14 No. 3, May-Jun. 2000, pp. 38-45.

[She98] Shehory, O., Sycara, K., Chalasani, P., Jha, S., "Agent Cloning: An Approach to Agent Mobility and Resource Allocation", IEEE Communications Vol. 36 No. 7, Jul. 1998, pp. 58-67.

[Ste98] Steen, M., Hauck, F. J., Homburg, P., Tanenbaum, A. S., "Locating Objects in Wide-Area Systems", IEEE Communications Vol. 36 No. 1, Jan. 1998, pp. 104-109.

[Ste99] Steen, M., Homburg, P., Tanenbaum , A. S; "Globe: A Wide-Area Distributed System", IEEE Concurrency Vol. 7 No. 1, Jan.-Mar. 1999, pp. 70-78.

[Tah82] Taha, H.; "Operations Research - an introduction 3ª ed."; MacMillan Publishing Co. Inc., 1982.