Evaluation of a large scale lookup algorithm in ASP based grids

L. Philippe S. Damy B. Herrmann I. Djama
S. Dahan
Laboratoire d'Informatique de l'Université de Franche-Comté (LIFC)
Route de Gray, 25030 Besançon cedex, France
contact: laurent.philippe@lifc.univ-fcomte.fr

Abstract

The Internet development and the availability of reliable networks led to the emergence of Grid architectures. The aim of these architectures is to take benefit of widely distributed resources to improve execution possibilities. Depending on their properties, these architectures are usually classified into desktop grids, resources grids and application based grids. Application based grids provide an easy access to applications deployed on the grid on the ASP (Application Service Provider) mode. When these grids grow of orders of magnitude, application lookup will become a costly activity of the grid. In this article, we study how a lookup algorithm scales when the size of the grid grows up. We exhibit a "lookup throughput" which characterizes the grid interconnections graph and the lookup algorithm.

Contents

List of Figures

Lookup algorithm, Application Service Provider, Grids, Graph traversal.

1 Introduction

The emergence of Grid platforms was allowed by the development of Internet and the availability of more reliable networks. The basic idea of grids is to provide computing resources on demand and applications everywhere. Currently grids are mainly dedicated to scientific applications. Several classes of Grid Platforms are distinguished. Desktop Grids like [2, 3] provide a framework to collect power from unexploited personal computers. Usually, these platforms are based on one, or more, servers which distribute the workload among idle clients. They provide execution resources at low cost but not all applications can benefit from this architecture. Only task independent applications are relevant. Resources Grids, as described by Ian Foster in [10], allow users to share execution resources. Thus, the Globus system [9] provides a set of servers which manage and allocate the resources to applications. This kind of platform is dedicated to the execution of parallel applications made of a set of tasks that communicate using a standard parallel communication layer as MPI. This means that only parallel programming experts can use these platforms and program the applications that will be executed. However, people may be interested in using the grid power without being experts in parallel programming or computer scientists. The basic idea of the applications based grids is that providers put applications at clients disposal. These applications run on powerful machines but are simple to access, as clients just have to send the parameters of their executions. This approach is normalized by the Global Grid Forum through the GridRPC [16] interface. This interface defines a simple RPC-like programming interface for accessing applications on a grid.

This application oriented approach is not limited to scientific applications and may be used by general purpose applications. In this case, the dedicated middleware is generally composed of three different roles: a service lookup, servers and clients. Examples of middleware platforms for scientific applications are DIET [6], a scalable Corba-based environment, Netsolve presented in [1, 19] and NINF described in [21]. In these platforms, the lookup function is centralized in an agent. This means that every client of the platform accesses the agent to choose the best available server. However, to provide access to a larger and geographically distributed set of services these platforms must be able to scale up. In the DIET case, scaling up is achieved by interconnecting several lookup agents in an overlay network. In this case, the middleware has to provide a distributed localization service or lookup service to find available servers. Localization is usually done by broadcasting search requests on the overlay network. However, broadcasting is very throughput consuming and must be controlled to avoid network overflow.

In this paper, we study the capability for a network of interconnected agents (overlay network) to support this localization service. In the first section, we introduce the model of application based grids. In the second section, we present the service lookup problematic, its application domain and related works. Then, we describe the algorithm used to broadcast service localization requests. In the forth part, we propose a model of the localization problem and we give details on its simulation using the SimGrid simulator. Finally, we present the results obtained when carrying out the simulations, analyze them and conclude.

2 Application Based Grids

The notion of service does not have the same meaning depending on the context. In distributed systems, services are applications provided to users through interfaces. In grid middleware, services are rather components offering system functionalities to applications executing on the grid. We use the term of application based grids to name the context of our work. Application Based Grids (ABG) define grid middlewares providing simple access to applications distributed on the computer grid. The GridRPC [15] model is an interface to application based grids. ABG are a tentative to merge Application Service Provider (ASP) with grid computing. Most of the application based grids lay on a three parts architecture, as shown on figure 1:

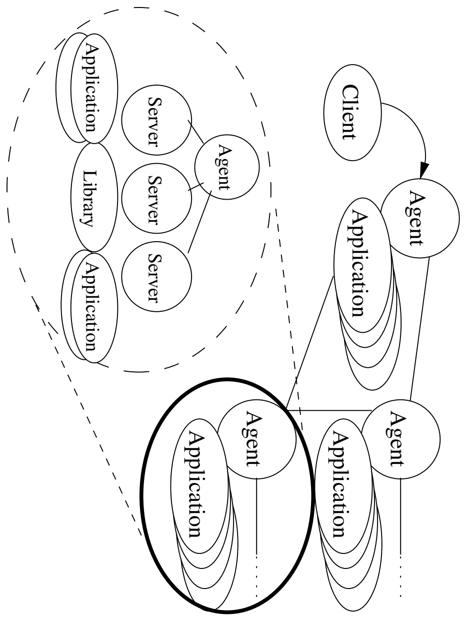

Figure 1. Architecture of ASP Grid

- the **server**, a Network Enabled Server or NES, provides a service. The server encapsulates an application or a library and makes it accessible by clients. Note that, in the following, the meaning associated with service is a type of application that is accessible through an interface, as in most distributed systems.
- the **client** accesses the grid to get results from an application with its own parameter set. It may be some kind of interactive tool or interpretor like Mathlab as well as a program. One of the advantages of ABG is that clients do not have to install large applications on their computers as they use them remotely.
- the lookup **agent**, is in charge of recording the services available on the grid. When a client looks for a service, it sends a request to the agent.

This architecture is common to other distributed computing environments. In fact, this is the common way for client to find servers in most Remote Procedure Call or Remote Method Invocation based platforms. However, the difference comes from the semantics associated with the lookup request. Some particularities of the ABG, compare to RPC or RMI platforms, are the size of the parameters and the resources consumed by one request. These particularities must be taken into account by the lookup agent.

Basic ABG implementations manage applications and servers in a campus or a company department. Applications and servers are made available to clients through an agent. Clients send their requests to this entry point which chooses or selects the best available server. This agent gives an access only to applications local to the department. So its application offers set is limited. By interconnecting different access points (agents), a larger set of services may be proposed to clients and the choice for executing one type of application will be made on a larger set of servers. In this case, agents must cooperate to implement a lookup function to find and select remote servers.

3 Scalable service lookup

To increase the number of services proposed, application based grids may be interconnected. We may assume that this interconnection will be extended worldwide one day. This means that the lookup service associated with a ABG must scale up. Several propositions exist in the domain of scalable lookup services and discovery algorithms.

The Corba Trader defines an interconnection graph to forward lookup requests in the trader federation as presented in [17]. To search for a service in the trader federation, the algorithm used broadcasts requests to the neighborhood until a defined Time To Leave (TTL) value is reached, on a Breadth First Search (BFS) mode. Other well-known distributed platforms provide lookup servers interconnection by adding references to other lookup agents as in Jini [20] or SDS [7] where administrators can interconnect lookup servers on a hierarchical scheme.

The lookup issue has also become popular thanks to the peer-to-peer (P2P) systems. In P2P systems, peers are looking for data or files stored somewhere on the network. Different lookup algorithms and architectures were tested on these networks. These algorithms may be classified as centralized or distributed. In centralized lookup algorithms, as Napster [18], location information are centralized on a server and the client accesses this server when he looks for a file. However, this does not scale well when the number of clients grows. Distributed algorithms try to avoid this bottleneck by distributing the location information among multiple nodes. Distributed algorithms are either based on super nodes sharing location information, as in FastTrack or gnutella [11, 14] or on data structures as Distributed Hash Tables DHT [13]. These algorithms lay on an infrastructure which has to be administrated or all the nodes do not play the same role in the lookup algorithm. The Gnutella P2P system used, in its early 0.4 version described in [5], an algorithm that is totally distributed on pairs. The lookup request was first sent to the neighbors of the initiating peer. Then, if the resource was not found on these peers, the request was sent to their neighbor peers, one hop further. Every wave of requests increments the hop count until it finds the resource. This algorithm led to a network overflow: as the network size has grown up, the number of lookup requests generated so much traffic that the peer network became overloaded. One of the solutions used to limit the lookup traffic was to limit the broadcast of requests using Time To Live (TTL). However this does not guaranty against a network overflow when the requests arrival rate is too high.

Service Based Grids are slightly different from peer networks. The request looks for an application and the resources which will be consumed by this application. When an execution may last for several hours, days or weeks, the execution server has to be carefully chosen. For this reason, the client request must be forwarded between agents in order to choose the server that best fits the execution needs. For instance, the server may proposes the best execution performances or security guaranties. In platforms like Diet or Netsolve, performance prediction is used to select the best offer before scheduling execution requests on servers. This selection implicitly balances the load between the servers. This selection makes the lookup process different from traditional lookup processes where the search is stopped when an instance is found.

4 Application Lookup algorithm

By interconnecting application based grids, we extend the set of application offers for the clients. Each ABG is locally managed and remains independent from the others. The interconnection of application based grid is done by a local administrator which declares the reference of other application based grid agents in the local agent, as in most lookup architectures. Then, the interconnection network of ABGs constitutes an overlay network above the Internet. The wider the ABG will be interconnected, the bigger the offers set will be and it may grow up to become world wide. The problem we face is how this interconnection network performs when the network size scales up? Probably, the load of lookup requests will grow and overloads the network as that is arisen for Gnutella. So what is the limit of requests arrival rate? How the network and algorithm parameters will affect the performances of the lookup process? These are some of the questions that we try to answer.

Of course, the answer depends on several parameters: the underlying network characteristics, the overlay network interconnections, the traversal algorithm used for service lookup, etc. As the set of parameters to be evaluated is too large, we had to set some. In this paper, we mainly focus on the overlay network characteristics. One of the characteristics we want to exhibit is that the overlay network may be characterized by some kind of throughput when a steady state is reached.

The algorithm used to look for services on the overlay network is based on the multi-waves broadcast algorithm already used by Gnutella. First, the client sends its request for an application to the agent it depends on. If the agent did not record a server providing this application, it forwards the request to other agents, on the overlay network. At a first step it sends the request to its direct neighbors that can be reached at one hop and waits for their responses. If none of the neighbors know a server for the application then the agent initiates a second wave of requests which queries the neighbors one hop further. The agent continues until it reaches a max-hop count or it finds a server providing this application.

To be sure that an agent is not asked twice for the same application, each request is identified by a request identifier and the neighbors of an agent are numbered. When an agent receives a request for the first time, it records the identifier and the number of the link it has received the request on, thus it records the emitter agent. Then, it looks up its local server repository for the application requested. If the application is present, the agent replies with a proposition to the broadcasting agent and the algorithm stops. If it is not available, the receiving agent replies to the sending agent that it does not provide the application. If the agent receives a request already recorded, that means it has already given a negative answer to this request. So, it forwards it to its neighbors if the hop count is under the Time-To-Leave value.

Several answers are possible to the forwarded request. The neighbors may have a server that provides the application. In this case, the lookup algorithm stops. The neighbors may not have any offer but have neighbors that have not been contacted, in this case the lookup algorithm forwards the request to these neighbors during the following wave. Last, the neighbors may not have any offer and have no non-contacted neighbors, in this case the lookup algorithm marks the agent has ended and indicates it to the forwarder agent by sending an end message. When an agent receives an end message, it marks its neighbor as ended and does not forward more requests to this neighbor as this part of the graph is completely traversed. When all the neighbors of an agent are ended, this means that this agent is ended and the algorithm sends an end message to the father agent. When all the neighbors of the broadcasting agent are ended, this means that the graph is completely traversed and, if no proposition is received, they are no offer for this application.

This multi-waves algorithm intends to limit the number of messages sent by an agent when a server provides the application in the neighborhood of the agent. It performs well if several instances of the application are available on the network but generates more messages than a simple broadcast algorithm when few servers provide this application on a wide network. Several improvements to this algorithm are possible mainly to limit the number of exchanged messages. However, our aim is rather to evaluate the overlay network capabilities than to provide the best algorithm.

To evaluate the behavior of the algorithm when the number of interconnected agent grows up, we model its context and simulate it.

5 Modelling

The agents interconnection network can be seen like a connected graph with non-oriented edges. The vertexes of the graph are the agents and the edges the links of the overlay network. Using this model, we evaluate the number of messages generated by a lookup request: first by maximizing it then by calculating an average messages number in case of a tree network.

5.1 Maximum messages number

The lookup algorithm widely broadcasts requests to find an application. The request arrival rate determines the performances of the algorithm as the network may become overloaded. This arrival rate depends on the number of messages generated by a lookup request. This number of messages depends on several parameters such as the maximum hop count used to forward a request, the number of neighbors of an agent and the number of applications in the network.

If the arrival rate is low, the load generated by requests does not disturb the network traffic. But when this arrival rate grows up, some congestion can occur and have an effect on the network performances. To evaluate these consequences, we maximize the number of messages generated by one request. The traversal of the graph using the lookup algorithm, without using waves nor TTL, generates $2 \times L$ messages (request and reply for each link), where L is the number of links.

However, when we introduce waves in the algorithm, the number of messages also depends on the graph topology. For instance on a star graph, where every vertex is connected to one central node, the number of messages is $2 \times (N - 1)$ messages, where N is the number of vertexes. Thus, in the case presented on figure 2, there are 24 messages (two per link) for one wave on 13 nodes. In the case presented on figure 3, there are 30 messages for two waves

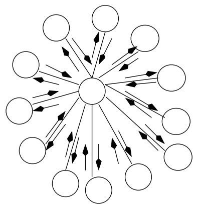

Figure 2. Message broadcasting on a star

on 13 nodes, 6 messages for the first wave and 24 for the second.

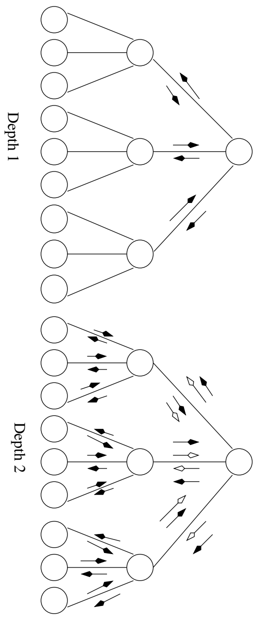

Figure 3. Message broadcasting on a tree

Let M_k be the number of messages generated by wave k. Thus, the maximum number of messages for a TTL of T is M_T.

5.2 Calculation of M_k :

To maximize the messages number, we assume that, from one node, the graph used is a tree, as shown on figure 3, to avoid accessing the same node twice. Then we compute the number of messages when the graph is completely traversed. The parameters used in this calculus are the following:

- d, the interconnection degree, is the average number of neighbors for an agent. This means that an agent broadcasts a request to d neighbors.
- T, the maximum lookup depth, is the maximum number of hops for a request. It is also the number of waves of the algorithm. The same value is used for all agents and requests.

The number of messages generated by one wave is two times the number of agents reached by this wave: for each emitted message an agent gets a response message. The first wave generates $2 \times d$ messages as the node has d neighbors. The second wave generates $2 \times d^2$. Thus, if the TTL is set to 2, the maximum number of messages is the number of messages generated by the first wave plus the number of messages generated by the second wave, $2d + 2d^2$. This can be generalized at wave k. The number V_k of messages needed to access all the nodes at k hops is:

$$V_k = 2\sum_{i=1}^{k} d^i$$

The number of messages generated by a request of depth k is the sum of the messages generated by the requests of waves 1 to k. Thus, the total number of messages generated by wave k is:

$$M_k = \sum_{i=1}^{k} 2\sum_{j=1}^{i} d^j$$

That can be simplified in:

$$M_k = \sum_{i=1}^{k} 2(k-(i-1))d^i$$

This result shows the maximum number of messages generated in this case. We can note that this value has an exponential progression on the TTL value. However, all the requests do not generate the maximum number of messages. We compute a mean number of messages depending on the probability to find an application on an agent.

5.3 Average messages number

Let X be the probability to find the application on an agent. This probability is modelled by a discrete random variable which may take two values: 1 if the application is found, otherwise 0. The probability to find an application is set to p, thus:

$P(X = 1) = p$
$P(X = 0) = q$ with $q = 1 - p$

We assume that the probability is the same one for all the agents. The average number of messages $\overline{N}$ generated by one request is:

$$\overline{N} = \sum_{k=1}^{T} P(N = M_k) M_k$$

Where $P(N = M_k)$ is the probability that the number of messages is equal to M_k. As M_k is a discrete variable, the probability is null when this variable is not defined.

When the graph is a tree, their are d^k agents at level k. So, the probability to find an application at level k is:

$(1 - q^{d^k})$

The probability to get M_k messages for a request of depth k is the probability to find the application at this level. It depends on the value of k compared to the maximum depth (TTL) T:

- for $k < T$: we did not find it at levels from 1 to $k - 1$ but we did at level k:

$$P(N = M_k) = (1 - q^{d^k}) \prod_{j=0}^{k-1} q^{d^j}$$

- for $k = T$: no application were found until level $T - 1$

$$P(N = M_k) = \prod_{j=0}^{k-1} q^{d^j}$$

Thus the average number of messages for a TTL of T is:

$$\overline{N} = \sum_{k=1}^{T} P(N = M_k) \sum_{i=1}^{k} 2(k - (i - 1))d^i$$

5.4 Example

In this example, we calculate the average number of messages generated by a request on figure 3, by using the previous result. The interconnection degree is $d = 3$, the max lookup depth is $T = 2$ and the probability to find the application on an agent is $p = 0.2$ $(q = 1 - p = 0.8)$.

The request may be satisfied either at a depth of one or two:

$$\overline{N} = \sum_{k=1}^{2} P(N = M_k) M_k$$

$$\Rightarrow \overline{N} = P(N = M_1)M_1 + P(N = M_2)M_2$$

The number of messages generated at level 1 is:

$$M_1 = \sum_{i=1}^{1} 2(1 - (i - 1))3^i \Rightarrow M_1 = 6$$

And the probability to get 6 messages is:

$$P(N = M_1) = (1 - (0.8)^{3^1}) \prod_{j=0}^{0} (0.8)^{3^j}$$

$$\Rightarrow P(N = M_1) = 0.488$$

The number of messages generated at level 2 is:

$$M_2 = \sum_{i=1}^{2} 2(2 - (i - 1))3^i \Rightarrow M_2 = 30$$

And the probability to get 30 messages is:

$$P(N = M_2) = \prod_{j=0}^{1} (0.8)^{3^j} \Rightarrow P(N = M_2) = 0.51$$

So, in this case, we have an average of 14.34 messages per request for a TTL of 2:

$$\overline{N} = 6 \times 0.488 + 30 \times 0.51 = 18.24$$

Using these results, we can calculate the average number of messages generated by one request and, with this number, we can calculate the consumption of resources in the agents network. For instance, we may calculate the network throughput or the CPU power consumed by the agents. Let assume an overlay network is composed of several agents interconnected with an average degree of 3. If all the agents have the same behavior then one agent will generate an average of 18.24 messages per request. Now, if we assume that an agent emit an average of 10 requests of 1 Kbyte per second, then the number of messages generated by one agent is 182 and the network consumption is 1.42 Mb/s just for application lookup!

However this result is an average value for the consumption of the resources and it does not take several parameters or configurations into account. The model used to get this result assumes that all the nodes have the same characteristics and the structure of the overlay network is regular (a tree). A more general model would lead to a too complex modelling. For instance, the given equation does not take into consideration links between nodes of the same depth. The use of these links will increase the number of messages for the same interconnection degree. Nevertheless, if we integrate these links in the equation, to be more precise, the equation will depend on the network structure and it will not be the same for all the nodes.

So, to better understand the behavior of the lookup algorithm we need to simulate it step by step. This allows the study of more realistic scenarios and exhibits the performances depending on the parameters.

6 Simulation

We want to exhibit through this simulation the limits of an overlay network and its dependence on parameters such as the interconnection degree of the graph or the maximum depth of request forwarding in the lookup algorithm.

To simulate the algorithm execution, we use the SimGrid simulator [4, 12]. This simulator provides a core of functions to simulate distributed applications in heterogeneous environments. These functions are used to develop a specific simulator. Some basic entities provided by SimGrid are hosts, tasks, processes and channels to exchange messages. Code may be assigned to processes to implement algorithms. A platform file gives all the characteristics of the platform used: cpu number and speed, links latency, throughput and interconnections. Our implementation is based on SimGrid 2.9.

In the case of the lookup algorithm, agents are simulated by processes executing the algorithm on a host. The interconnections between agents, the overlay network, are simulated by links. The requests are sent through messages on these links. When an agent receives a request it activates a task to handle it.

Applications are recorded in agents. Several servers which provide the same application may be recorded in the same agent. So the agents just record the different application types as application offers. Before the simulation the application offers are statically distributed among the agents. Then, on a lookup request, an agent searches in its repository if it recorded this application. If the offer is present, then the agent creates an evaluation task. This task simulates an evaluation of the application offer as execution performances of the server or properties checking.

The SimGrid simulator uses a platform description file to generate the links between hosts. This platform file gives a list of every existing link between nodes. Because the size of the interconnection graphs is too big to be defined by hand when the number of nodes exceeds few hundreds and to generalize our results on different graphs, the interconnection definition is based on general characteristics and, thus, it is not limited to one particular graph. Characteristics we are interested in are the number of hosts, the interconnection degree of the hosts and the type of graph (random, grid, shuffle-exchange, etc.). To generate graphs depending on these characteristics, we developed a generator which produces graphs depending on properties given in parameters. This graph generator allows to define links properties: throughput and latency. All the links have the same properties. The same configuration file sets the hosts (agents) properties as cpu speed, duration to process and evaluate a request.

As said before, using the defined graph, we distribute the application offers among agents. As the size of the graph may grow up to several hundreds of hosts, this distribution cannot be done by hand and we use a statistic distribution. This distribution depends on two parameters: the number of different application types and the number of application instances per type.

The last parameters given for a simulation define the lookup algorithm and the requests properties. For the lookup algorithm, the maximal depth for stopping a broadcast (TTL) and the maximum number of offers returned on a request can be given. For the requests, the properties are the number of requests handled during a simulation, their distribution among the hosts and their arrival rate law (random or uniform).

The result of the simulation is observed through output values. These output values are periodically captured and stored in results files. The observed values are:

- number of requests at a time: this value gives the load of the agents network according to time.
- number of researches at a time: this value gives the number of requests that are in the evaluation process of the hosts. These requests do not use the network for the moment but they will do it as soon as they will be finished with the evaluation process.
- number of messages at a time: this value gives the number of messages that are currently on the links. This gives a accurate value on the current network load .
- number of messages per request: this value gives the number of messages generated by each request.
- duration of a request: the time between the creation of the request on a host and its termination.
- ratio of satisfied requests: this value is expressed in percentage. As some requests may not find an application offer corresponding to their request, this value gives the ratio between satisfied requests and not satisfied requests.

7 Results

In this part, we present results we got with different simulations. The first characteristic we exhibit is the idea of "lookup throughput". By lookup throughput, we mean the number of requests managed by the agents or rather the request arrival rate supported by the network without being overloaded.

7.1 Lookup throughput

Figure 4 shows the number of requests in the agent network for different arrival rates ranging from 1.5 ms to 3 ms (mean time between two requests) in a network of 100 agents. The links throughput is set to 1 Mb/s and the cpu consumption in an agent for one request is 1 ms. There is 100 offers available in the agent network with 10 different

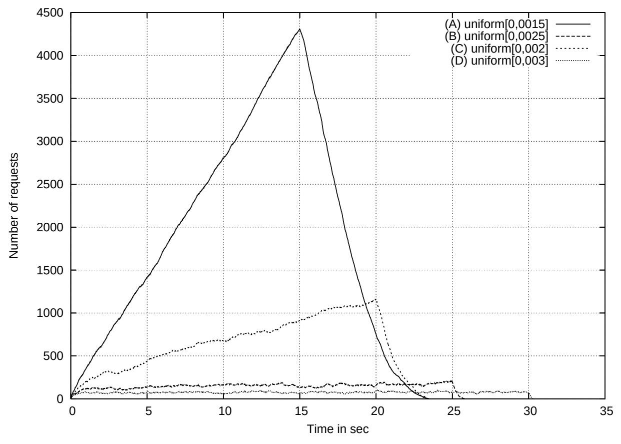

Figure 4. Number of requests (1)

types of application. Thus, on very agent, the probability of finding an application is 0.1.

On the figure it is rather obvious that an arrival rate of 1.5 ms will saturate the agents network and that an arrival rate lower than 2.5 ms can be easily supported. However, for arrival rates between these two values more details are needed. For instance, when the arrival rate is set to 2 ms, the curve slowly goes up which seems to indicate a saturation of the network . To define more precisely the threshold of saturation, we used a shorter range of arrival rate variations.

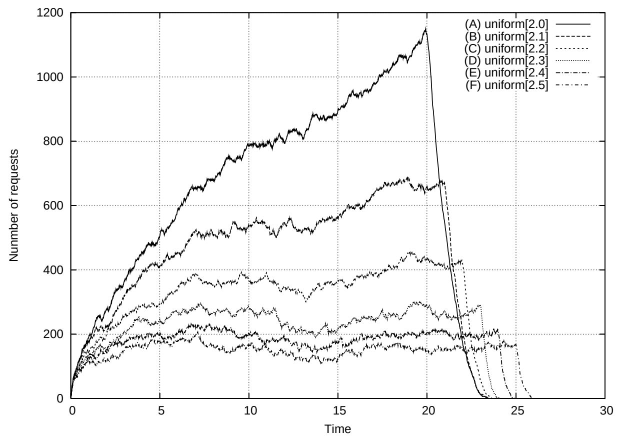

Figure 5. Number of requests (2)

Figure 5 gives more precise indications on the network saturation for uniform arrival rates ranging from 2 ms to 2.5 ms. On this figure the difference between saturated and unsaturated curves is less obvious. We can note that, in a first phase, the number of requests grows. Then, according to the arrival rates, the value seems to stabilize for arrival rates greater or equal to 2.2 ms. For 2 ms, the value indicates a saturation but for 2.1 ms it is not possible to decide if there is a saturation or not: the network is in an unstable state. Since the network goes through an unstable state between arrival rates that do not saturate the network and arrival rates that clearly overload it, we can conclude that defining a metric to precisely set the lookup throughput is mandatory.

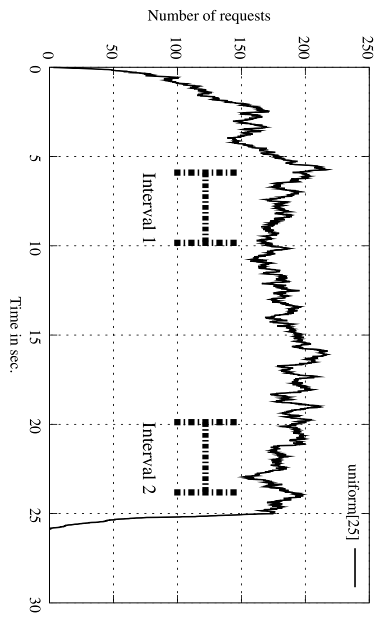

Figure 6. Intervals for stability

To characterize the lookup throughput we need to define a criterion based on the number of requests, such as this lookup throughput is defined by the maximum threshold where the arrival rate does not saturate the agents network. As the requests number is dynamic and not predictable, there is no absolute threshold that determines if the value is stable or not. So, we have defined a relative criterion based on the evolution of the requests number. We take two intervals in the simulation, as shown on figure 6. The first one is taken at the beginning, after the initialization phase when the network is not filled by requests, and the second is taken at the end, before the finalization phase where no more requests arrive. We compute the average number of requests in these two intervals. If the difference of these average values is less than a threshold, the network is considered as stable. It is considered as unstable otherwise.

7.2 Network accumulation

To have a better understanding of the reasons of the network accumulation, we have identified the factors that influence the performances of requests processing. The requests

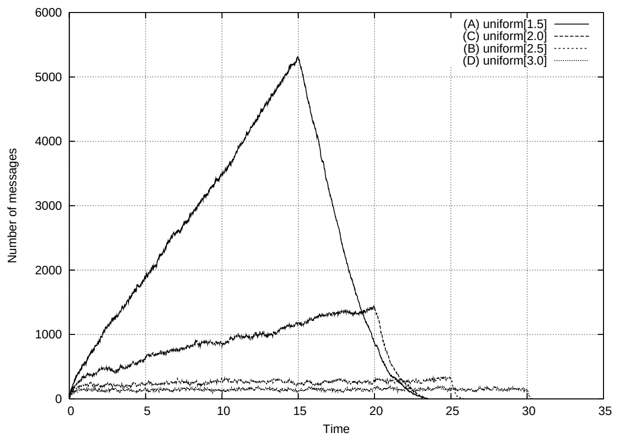

Figure 7. Number of messages

completion depends on the resources used by a lookup request: the underlying physical network throughput when the messages circulate on this network, the execution time in agents and the evaluation time spend in servers. We vary these parameters to show their links with request completion performances.

First, we have studied the number of messages in the agents network on figure 7. This figure is very similar to figure 4. This indicates that, at a given time, most of the lookup requests are contained in messages circulating on the network. So, the accumulation of requests is actually in messages on network links. This was also confirmed by observing the number of on going evaluations which is stable over this time. These curves show clearly that the physical network is the bottleneck of the lookup algorithm.

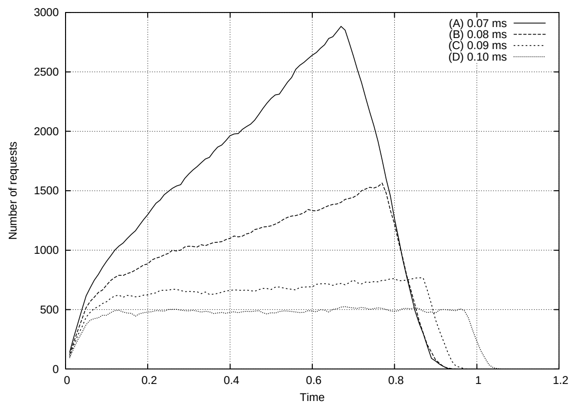

Figure 8. Requests with no network cost

To verify this result, we have changed the links parameters to suppress the message exchange cost. In figure 8, the network latency was suppressed and the link throughput (physical network) was set to 100 000 Mbs. On this figure, we can note that the agents network throughput raises significantly up as a request arrival rate of 0.09 ms does not overload the network. This proves that the physical network throughput is the critical resource needed by the lookup algorithm.

Note that another accumulation of request arise in the agents network for arrival rates above 0.09 ms. This accumulation is due to agents processing of requests and the limited computing resource: a processor is not infinitely fast. The mean cpu consumption time is set to 1 ms per request. This was checked by giving no cost to request processing. In this case, the number of requests in the agent network is constant over the time and directly depends on the request arrival rate.

7.3 Influence of network throughput

To characterize the influence of the physical network throughput on the agents network capacity, we determined the request arrival rate which verifies a stable state (using the criterion introduced in section 7.1). This request arrival rate was determined for a given number of agents (100, 500, 1000, 1500, and 2000) and for different network throughputs. The results are given in figure 9. On these curves we can see that the number of requests in the agents network linearly depends on the network throughput. For example, with 2000 agents, when the network throughput increases from 50 Kbs to 0.5 Mbs, the number of requests increases from 5000 to 50000. The figure also shows that the lookup throughput linearly depends on the agents number for the tested network throughputs. This means that the lookup algorithm scales well in this context. Nevertheless, it is important to note that all the experiments were done with a lookup depth of 2 and a probability to find an application of 0.5 and that the results are highly dependent on these values. It is obvious that, as the lookup depth increases, the number of requests raises up and that this number raises down when we increase the probability to find an application.

An important remark is that the first measurements were done with 1 Mbs of network throughput. The results show that, in this context, an agent is able to handle about 4.5 requests per second without overloading the network. However, this means that, to handle 4.5 requests per second per agent, we use 1 Mb of network throughput that is rather large just for applications lookup. This result does not correspond to the one given by theoretical calculation in 5.4. All reasons that may be invoked show the difference between theory and practice: the network is here not regular and the computing time on agents has not been taken into account.

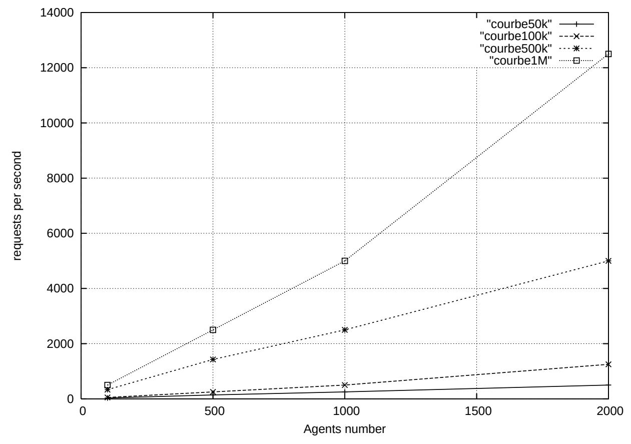

Figure 9. Network throughput

7.4 Influence of Time-To-Leave value

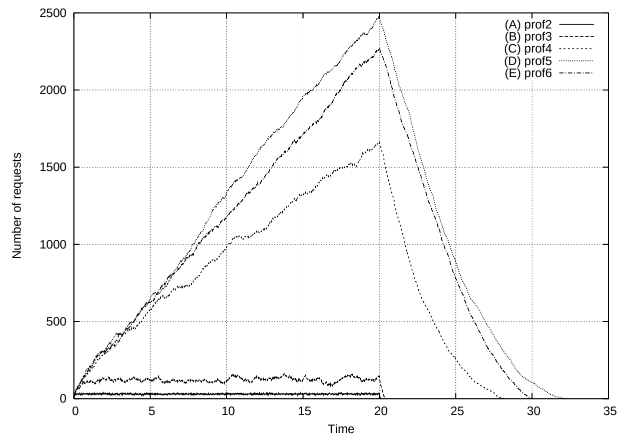

Figure 10. Requests Number with TTL

In the previous results, the TTL value was set to 3. The TTL value sets the hop count, or depth, where messages are not more forwarded to neighbors. It is obvious that this parameter has a great influence on the network load as the number of messages generated by one request depends on it. Figure 10, gives the results obtained when TTL varies from 2 to 6 in a hundred nodes network. Note that on a tree with an interconnection degree of 3, a request reaches at most 81 (3^4) nodes with a TTL of 4 and all the network with a TTL of 5 ($3^5 = 243$). For this reason, curves for a depth 5 and 6 are very close. The difference between curves (B) and (C) confirms the influence of the TTL value. In this experiment the probability to find an application is 0.2. In this case, the agent has a good probability to find an application offer in two waves. However, TTL above 2 have an influence on the load of the agents network. This can be explained by the random choice of the graphs. Other experiments done with a probability of 0.5 shows that the application offer is found without generating much messages and the agents network is not overloaded whatever TTL we use.

8 Conclusion

In this article, we study the throughput of a network of agents for a lookup algorithm. The context and properties of the lookup algorithm are slightly different of classical lookup algorithm. Our context is the context of application based grids where clients look for application types and performances. In this context, applications cannot be simply recorded in a centralized repository for two reasons. First, applications are structured by administrative domains where people need to keep control on the applications use. Second, clients choose the executing server for the application according to an evaluation of their performances. This performance evaluation needs to access the server to use dynamic data.

The algorithm used is based on a simple multi-waves broadcast. We did not have the intention of finding the best algorithm but we rather wanted to test the network of agents capability to handle requests. First, we give the theoretical limit for the lookup network throughput by calculating the maximum number of messages generated by one request. But the result is complex and is limited to regular networks as trees. Then, we simulated the network and pointed out that the lookup throughput mainly depends on the physical network throughput rather than on the execution time on each node and that the lookup algorithm is very network consuming.

Note that, all these results may also be applied to certain classes of peer to peer systems for requests broadcasting. In the case where pair may not look for resources available in the network but rather broadcast administrative informations. In both cases, the underlying network is an overlay network. Our results give interesting information on the cost and the limited capabilities of an overlay network to support broadcast.

To follow this work, we are improving the algorithm performances by implementing offer caches that record previously found offers on agents. An other improvement of the algorithm is based on the use of proxy-offers broadcasted by agents as described in the Corba Trader specification. We also plan to evaluate more parameters as the overlay network structure. Finally, this example will be used as a comparison base for different broadcast algorithms and structures as described in [8].

References

[1] S. Agrawal, J. Dongarra, K. Seymour, and S. Vadhiyar. *Grid Computing – Making the Global Infrastructure a Reality*, chapter 24. NetSolve: past, present, and future; a look at a Grid enabled server. John Wiley and Sons, Ltd, 2002.

[2] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, Nov. 2004.

[3] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, N. Vincent, and O. Lodygensky. Computing on large scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *FGCS Future Generation Computer Science*, 2004.

[4] H. Casanova, A. Legrand, and L. Marchal. Scheduling distributed applications: the simgrid simulation framework. In *Proceedings of the 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, page 138, 2003.

[5] Clip2. The gnutella protocol specification v0.4. Technical report, clip, 2001.

[6] P. Combes, F. Lombard, M. Quinson, and F. Suter. A Scalable Approach to Network Enabled Servers. In A. Jean-Marie, editor, *Advances in Computing Science - ASIAN 2002. Internet Computing and Modeling, Grid Computing, Peer-to-Peer Computing, and Cluster Computing. Seventh Asian Computing Science Conference*, pages 110–124, Hano, Vietnam, Dec. 2002. Springer-Verlag. LNCS 2550.

[7] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Proceedings of the fifth international conference on Mobile computing and networking*, pages 24 – 35, Seattle, Aug 1999. ACM Press.

[8] S. Dahan, J.-M. Nicod, and L. Philippe. The Distributed Spanning Tree: A scalable interconnection topology for efficient and equitable traversal. In *5th Int. Symposium on Cluster Computing and the Grid (CCGrid 2005), workshop on Global and Peer-to-Peer Computing, GP2PC 2005*, Cardiff, UK, May 2005. IEEE Press. CD-ROM.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

[10] I. Foster and C. Kesselman, editors. *The Grid, Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1999.

[11] T. Hargreaves. The fasttrack protocol. July 2004.

[12] INRIA. Simgrid. electronic. `http://simgrid.gforge.inria.fr/`.

[13] E. Keong Lua, J. Crowcroft, M. Piaz, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Survey and Tutorial*, 7(2), Apr. 2005.

[14] T. Klingberg and R. Manfredi. Gnutella 0.6, June 2002.

[15] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. GridRPC, a remote procedure call API for grid computing. Technical Report ICL-UT-02-06, Innovtive Computing Laboratory Report, June 2002.

[16] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. A gridrpc model and api for end-users applications. Technical Report GFD.52, Global Grid Forum, http://www.ggf.org/gf/docs, 2005.

[17] OMG, http://www.omg.org. *Trading Object Service Specification*, 97. orbos/97-07-26.

[18] S. Saroiu, K. P. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems*, 9(2):170–184, Aug. 2003.

[19] K. Seymour, A. YarKhan, S. Agrawal, and J. Dongarra. Netsolve: Grid enabling scientific computing environments. *Grid Computing and New Frontiers of High Performance Processing*, 2005.

[20] SUNLABS. Jini architecture overview. Technical report, Jan. 1999.

[21] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G: A reference implementation of RPC-based programming middleware for grid computing. *Journal of Grid Computing*, 1(1):41–51, June 2003.