Some Solutions for Peer-to-Peer Global Computing

Guillaume Jourjon, Didier El Baz
LAAS du CNRS
7, avenue du Colonel Roche, 31077 Toulouse Cedex 4, France

Abstract

the emergence of Internet and new kind of architecture, like peer-to-peer (P2P) networks, provides great hope for distributed computation. However, the combination of the world of systems and the world of networking cannot be done as a simple melting of the existing solutions of each side. For example, it is quite obvious that one cannot use synchronized algorithms for global computing over large area network. We propose here a non-exhaustive view of problems one could meet when he aims at building P2P architecture for global computing systems, which use asynchronous iterative algorithms. We also propose generic solutions for particular problems linked to both computing and networking sides. These problems involve the initialization of the computation (and its dual the conclusion), the task transparency over P2P network, and the routing in such networks. Finally a first computational experiment is presented for an asynchronous auction algorithm applied to the solution of the shortest path problem.

1 Introduction

Distributed computing allows searchers to simulate their theory as well as it allows solution of large-scale problems. Many numerical problems can generally be seen as fixed-point problems. These problems can be sometimes solved thanks to iterative methods. Furthermore iterative methods can be divided into two different groups, synchronous and asynchronous. For thirty years computer scientists and mathematicians, see [2, 5, 4] and [14], explore the way to study and implement these algorithms over distributed and parallel architectures, but with the emergence of networks for the past twenty years a new field of experimentation is available.
After the client/server architecture, new network architecture is born, the P2P network. This kind of networks differs from the previous, and dominant, architecture by the fact that every computer in the network plays the same role. The first use of this wide scale architecture was a file sharing system with the success of Napster, but we can notice that this application was just the first generation of P2P networks, because it used a centralized architecture. Another success of first generation P2P systems is the project of the Berkeley University, Seti@home
These P2P first generation applications still suffer by their centralized architecture and lack of communication between peers, e.g. in Seti there is no communication between peers. The second generation of P2P networks provides a pure decentralized architecture with no central server, which connects peers. The first and only available type of applications for this generation is file-sharing systems like Freenet or Gnutella [13].
The performances measured over such Networks (65 Teraflops/sec for Seti) let presume a great capability for computing. Furthermore the improvement done in communication process (Cable...) will allow an implementation of global computing algorithms.
The goal of our work is to set out an overview of the asynchronous iterative algorithms to solve great problems and to exhibit the main problems one can encounter for the implementation of such schemes over a P2P network and the way to solve several of these problems. Section 2 presents gen-

eral asynchronous iterative algorithm. The next section presents the main problems such a project has to solve, and sections 4, 5 present solutions for the problems of routing and initial repartition respectively. Eventually the section 6 proposes a general design for such a system, and gives first results obtained thanks to the implementation of our solution for the shortest path problem solved via an asynchronous auction algorithm.

2 Parallel Asynchronous Iterations

In this section, we give a brief presentation of parallel asynchronous iterative algorithms. We only present classical parallel asynchronous iterative schemes (see [2, 5, 8, 14]), but there is possible extension to more general schemes called flexible asynchronous iterations (see [9, 12]).
Parallel asynchronous iterative algorithms are generally used in order to compute an approximate solution to the following fixed-point problem $x^* = F(x^*)$, where x^** is a solution vector of $\mathbb{R}^n$ and F is a given fixed point mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$. Let $E = \mathbb{R}^n$, the following partition of E is made: $E = \sum Ei$, for $i = 1$ to p, where p is a given natural number which is relevant to the number of processors (peers). Accordingly, the iterate vector is partitioned as follows: $x = x_1, \ldots, x_p$ where each x_i is called a block-component. Each block-component x_i is assigned to a given peer. The fixed-point mapping is decomposed similarly: $F = F_1, \ldots, F_p$. Asynchronous iterative algorithms generate successive approximations of the block-components of the iterate vector x. These approximations are obtained by applying in parallel and repetitively the block-components of the fixed-point mapping F, to a given initial approximation $x(0)$. A simple definition of parallel asynchronous iterative algorithms can be given as follows.

Definition 2.1 *a parallel asynchronous iterative algorithm is a successive approximation method whereby several processors cooperate via data exchange to the solution of a given problem, and each processor performs computations, i.e. iterations, at his own pace by using the last available updates.*

As a consequence, parallel asynchronous iterative algorithms are methods whereby iterations are carried out in parallel by several processors in arbitrary order and without any synchronization. The restrictions imposed on parallel asynchronous iterative algorithms are very weak: no component or block-component of the iterate vector must be abandoned forever and more and more recent updates of the components have to be used as the computation progresses. The advantages of asynchronous iterative algorithms are computation flexibility, tolerance to problem data changes (the algorithm adapts itself to a modified environment) and fault tolerance (the algorithm works out even if some data are lost).
We present now a simple illustration of parallel asynchronous iterative algorithms.

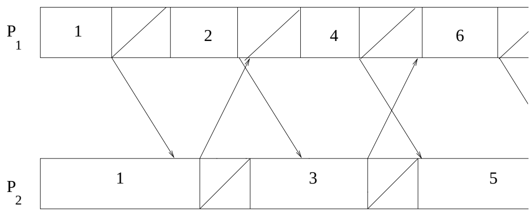

Figure 1: Asynchronous iteration

The above figure displays the behavior of a typical parallel asynchronous iterative algorithm in the simple case where two processors cooperate to the same application, i.e. the solution of a given fixed-point problem. The updating phases of the different processors are represented by boxes and the communications by arrows. We note that there is no idle time.
We will not give more precisions about this kind of parallel algorithm, for further information we refer to [2, 5, 9, 8, 10, 12, 14].
In the next section we will present specific problems concerning the integration of asynchronous iterations over a P2P network.

3 Presentation of the Problems

Problems can be, in a first time, seen as independent ones. We will enumerate a non-exhaustive list of problems one generally has to solve in P2P system, and in a global computing system. Each of these problems can have different definitions across literature, that's why we have to clearly establish definition of the terms we will use all along the study. More specific definitions of the problems we treat can be found in the sequel.
We will give a practical explanation and implementation of these properties. Let's first define the network's properties.
The scalability of a P2P network is often described as the principal interest of such a network. However, a unique definition of it is rarely given. Indeed, the definition of such a term depends on the scientist's approach of his own system.

Definition 3.1 *he scalability of a P2P network designed for global computing is its capacity to maintain its efficiencies when peers join or leave the system*

Efficiencies of a system of global computing over P2P network are numerous. Its is the routing efficiency, the search effectiveness, the algorithm's speed ...
Each of these efficiencies can be represented and measured with the help of numerical parameters based on different times and costs. We will see in the next sections how this property impacts in the setting of the system.
The second definition, we have to fix out, is the task transparency of a P2P network. We first have to separate the term of transparency and the one of anonymity, because in the literature these two concepts are often dangerously mixed up. Indeed, the concept of anonymity has to be linked with cryptology, which cannot be assimilated as a consequence of the network topology.
If we look at the file sharing P2P system we can define more precisely what should be transparency in our system. Indeed, in such applications the transparency is equivalent to a local or remote access to each block of a file.

Definition 3.2 *the transparency can be defined as the property to make undistinguished local or remote access to all parts of the task and data set needed for computation.*

The definition means that, whatever happens to the network, each peer still online can have access to the entire needed set of components for the computation. This can be translated by the fact that we need to envision duplication and a good distribution of this set of data and tasks. But if we compare to what is done in sharing file P2P system, the transparency just means that the system has to have an efficient search mechanism. (see [6])
Eventually, the last network's property, we have to define, is the robustness. The robustness, in a general overview, is the capacity of any system to keep on stability when an error occurs. The errors in a P2P network are the failures of peers and links between them. These failures can occur for many reasons, one can be attacked, another can just leave the network, or one could have his first IP router congested? If we want to model this with the help of graph theory, an error can be represented by the eviction of a node and all of its incoming and outgoing edge on the representative board, or just the eviction of an edge.

Definition 3.3 *the robustness of a P2P network is its capacity to stabilize itself despite the failure of some of its components (peers and links).*

This last definition is not only a network's definition, but it is also a computational one. In the following section we will describe two kinds of routing, which are needed in a P2P Global Computing.

4 Routing in a P2P network

In order to solve the different problems we have just presented, we need to set on, in a P2P Global Computing system, two kinds of routing. These two kinds of routing differ one from another in their goal and use. Indeed, the first routing that we will describe later aims at informing peers about the system's topology. That's why we name it the initial routing. The second routing aims at allowing efficient searches. It is linked to the transparency problem because as we have already said the transparency can be set on thanks to an efficient search scheme. This particular routing is the basic routing of a P2P system, as described in [6, 7]. Because of the difference in the way they are used, one routing is a search routing and the other is a finding host routing, these routings may, in a first time, be separate in order to keep on clear. But we can easily imagine that someone using any mathematical formula could combine these two kinds of routing in order to keep only one. But we are not sure that it is very useful.

4.1 Initial routing

The routing for the initial repartition of the algorithm aims at serving at the global efficiency of the algorithm. In this way we can think about an algorithm that defines a metric, which should be calculated thanks to peers' capacity, efficiency, and availability. These pieces of information could be calculated as an average of all the operation a computer can do per second, for example.
Thus, such algorithms are quite complicated and for the moment we just want to set on a simple routing algorithm mapped on the IP routing experimentations. That's why, in a first time, we recommend the use of basic algorithm. We can choose a Hop-count algorithm like RIP (see RFC num 2132) and we have translated it for P2P network. As a result, each peer owns a routing table, which contains a list of known peers, the next peer to send data in order to reach another peer, and the number of peers data has to go through before eventually reaching the wanted peer (hop count).
We will illustrate in section five how one can use this routing thanks to some algorithms.

4.2 Transparency routing

In this subsection, we will discuss about another routing, which differs from the previous one by its use and information it offers to peers. This routing aims at allowing peers to proceed efficient searches over the network. It doesn't certainly aim to replace the initial routing; it has to be seen as a complementary routing.
One of the main differences between this routing and the previous one is that this routing could, in the case there is no computation on the network, be useless.
The literature proposes several type of routing, which differ one from another following these hypothesis:

***Hypothesis* 4.1** *free form search criteria.*

***Hypothesis* 4.2** *search by key words*

***Hypothesis* 4.3** *search thanks to a distributed hash table. (DHT)*

First, we have to notice that the search in our system is a key words routing (hypothesis 2). Indeed, when a peer would want to search for a part of the data set and tasks needed for the computation, it has to know the name of the problem the set depends on. The fact that we can use name of problems is a quite important advantage for the search mechanism, because it allows more efficient search as compared to flooding search schemes of P2P systems like Gnutella. In order to establish such a kind of routing, we can use several solutions. We will use, for example, a transposition of one of them proposed by H. Garcia-Molina, which is called Routing Indices (RI)[7]. The difference between our solution and Garcia-Molina's one is that our solution locates part of the task and data set instead of the entire document.
Routing table for transparency will be established by using the same messages as the initial routing, or by using new messages that we have to add to the system's protocol.
The previous table represents the routing table for

	PJ722	KZ84D	XW583d
Peer A	20	50	20
Peer B	30	40	10
PeerC	10	30	60

Table 1: Routing table of a peer D

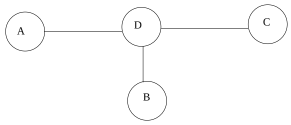

Figure 2: Associated network

peer D in the above network. The first column contains the identification of D's neighbor, and the three other columns are linked to three problems, the system is dealing about. In each problem's column, the number, in front of peers' identifier, represents the metric associated to the number of part of the task and data set you could reach if you send a query to the appropriated neighbor. This metric could be defined in many ways, for further information we refer to [7].
The search routing we have just presented is just an example of this kind of needed routing. We can also set on, for example, a routing that would take advantage of the generally encountered topology. P2P network's topology follows, like Internet and many other networks, power-law [1, 11]. That's why we could establish a routing that aims at finding network's super-nodes. Indeed, super-nodes are peers, which own more neighbors than other peers. As its possess many neighbors, a super-node also possesses more information for a search. However we think that in a first time search routing has to be quite simple. Because we first have to simulate and establish simple routing in order to guarantee that global computing is possible over a P2P network. Once we will have demonstrated that it is possible, then we would think about a more efficient routing which would need more computational time and cost.
We have established two routing, which are the base of a P2P Global Computing system. Now we will propose solutions for two main problems of a P2P global computing system.

5 Initialization and updating

In this section we propose simple solution for initialmisation and transfert of the solution.

5.1 Initialization

In this subsection, we present a mechanism, we call initialization, that aims at allowing the system so that it owns several properties that would be useful for the computational algorithm. Indeed, if one would want to solve a large-scale problem thanks to an asynchronous iterative algorithm, he has to envision the termination's detection. Thanks to the different schemes we propose here, and the scheme proposes in [8], we could easily assure the termination's detection. We have to notice that this partition will be achieved thanks to a unique message that we will call itSetup. This message will differ relatively on the algorithm of partition. For example if one uses the decideAll algorithm, the itSetup message will carry the identification of the root, the identification of the destination, the identification of the next peers the message would be forwarded to, the initial vector X0, and a piece of information (task and data set) that is assigned to the destination peer.
In order to study the initialization we suppose that in a first time the network is static.
We have found three simple methods to do this initial partition :

- The peer, which initializes the computation or root, splits the fixed-point problem into n parts, where n is the number of peers it could reach thanks to is routing table. Then, it sends these parts to peers by name. We call this method the decideAll method.

- The root splits the fixed-point problem into d parts, where d is its number of neighbors. These d parts could be different in size, according to the number of peers reachable by each neighbor. Then, it sends these parts on, and at the neighbor level the algorithm restarts (this method is called diffusion).

We will see in the sequel a third method that we call the evaluateAndSend one. We first discuss about the advantages of the two previous methods.
The main advantage of the first method lays in the fact that root, namely the peer that gets all the information, decides of the partition. On the other hand

since the network is dynamic, it is possible that the network topology changes, hence it is possible that some peers are no longer reachable by the anticipated path.
The diffusion method owns the advantage of repartition closer to the network topology. But we can also notice that this point may be a drawback if the topology possesses cycles.
The third method we propose in this study is called evaluateAndSend, and should be squared as follow. The root splits the fixed-point problem into m parts, where m is the result of an algorithmic evaluation of the number of peers. This algorithm could be sophisticated as those proposed in [3], or simple as the one used in the decideAll method. After this evaluation, the root splits the problem into m parts then, it sends these parts to its neighbors, each neighbor keeps the received part and processes it if the peer is idle, or keeps the received part and forwards it to one of its neighbor (different from the neighbor which sends it previously) otherwise.
We propose now three pseudo-codes for these three algorithms.

```
decideAll:
input: fixedPoint F, originVector V
output: order for beginning
computation
Int n = CalculateNumberPeer (
               Routing Table)
F[n] = SplitPb(  F)

for each peer[i], i from 0 to n-2{
send(peer[i], appropriatedNeighbor,
               F[i], V)
}
begincomputation(F[n-1])
```

Table 2: algorithm decideAll

```
diffusion :
input and output unchanged
Int d = numberOfNeighbors
//evaluation of the neighbors
map mapNeighbors = evalNeighbor(d)
//this map should contains a key
for each neighbor and as value a
real number between 0 and 1
F[d] = SplitPb( mapNeighbors, F)

for each neighbor i
send( i, F[i])
// each neighbor repeats this
algorithm at the reception of the
message itSetup
```

Table 3: algorithm diffusion

We can notice that each method we have proposed forwards the message itSetup by using the topology of the network. This is an important feature of the initialization, since thanks to this feature we can build a tree, which is very important for the termination of the asynchronous algorithm. Indeed, we could use in order to detect the termination of the computation the principle presented in [10].
We show now how one peer can perform update of an iterative algorithm.

5.2 Updating

In this subsection, we study updating processes for asynchronous iterative algorithms over P2P network. Definition of asynchronous iterative algorithms compels updating processes to follow rules that could be translated by the fact that moreover the computation goes on, the values of iterate vector's components used in the updating processes have to be more and more recent and any iterate vector's component has to be regularly updated. In order to insure these properties, we propose two solutions. Each solution uses a message, which we will call itNew (iteration end), that will be used to communicate the update of the vector.
The first solution we propose here consists in the propagation of this message to the entire network. We could easily imagine that this solution will have a problem, which is the communication cost. Indeed

```
evaluationAndSend:
input and output unchanged

int n = evalNumberOfPeer
// algorithm one can choose
 among several
known count algorithm
F[d] = SplitPb (n)
for each peer[i], i from 0 to n-2{
send(appropriatedNeighbor, F[i],)
}
```

Table 4: algorithm evaluateAndSend

if the vector owns a great number of components the message itNew could be quite big. Even so we can notice that cycles in the propagation of this message could be avoided with a simple test, which could be summarized to a backup of the message on each peer. This solution suffers from the fact that broadcasting is often useless, since in the resolution of large scale problems dependencies between vector's components are not global, especially in distributed systems.

The second solution consists in a shorter propagation of this message, but in order to insure that each block of the iterate vector is still in updating, we propose that in the initiation phase each peer is assigned to regularly control the ongoing updates of a components' block of the iterate vector. We propose to accomplish it the following schemes:

- Every t seconds each peer verifies that the components' block it is in charge has been updated since the last verification
- If this block has been updated the peer proceeds in its duty, otherwise it sends a query to the system in order to know if the peer that updates this block is still in computation.
- If the searched peer is still on the network, it become a neighbor of the peer, which sends the query, otherwise the first peer will search for the data set and task needed for the computation of this block and it will replace the missed peer.

This solution owns two advantages. The first one is its lower communication cost. And the second is its improvement of the robustness of the system. Indeed with this solution we guarantee the detection of the non-updating of a block of component, and in response to this detection we could set on several counter measures, as an automatic search of the associated task and data in order to compute it in another peer.

6 Design of a first solution

After seeing all these solutions we propose in this section a first global architecture for a P2P Global Computing system. We will enumerate a list of needed messages and their role in such a system. The messages we speak about in this solution are as well messages like in the Gnutella protocol [13] or remote method if we use the middleware technology.

6.1 General Design

First of all, in P2P networks we have to maintain the topology. In this goal, we have to envision a pair of messages that aims to verify that the neighborhood of each peer still belongs to the network. We call these two messages ping and pong. The ping message is a kind of question that other peer will answer by sending a pong message. Ping message doesn't contain anything, but pong message could contain routing information if the routing table of the questioned peer has changed since the last ping received message. We can illustrate this scheme as follow. The

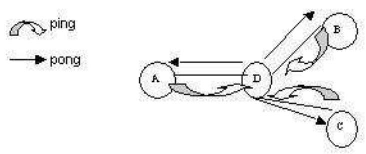

Figure 3: Maintaining topology via ping/ping

ping message could, in certain case, be used for state of Peers notification (active, inactive, terminated) as

proposed in [10].
The second class of messages we will specify is about the algorithm ongoing. We have already spoken about two messages needed for asynchronous iterative algorithms, itSetup and itNew.
The message itSetup always owns in its payload the origin vector that peers will update, and part of fixed-point mapping that allows peers to update a block of component of the vector. This needed payload could be easily carried out thanks to the XML format, especially for the vector. This payload could, as we have already spoken about, contain more information, like the destination of the message, the source ...
The message itNew owns in its payload the iterate vector or a part of it, depending on the problem and on the updating scheme we could set on. This message will also possess a field called TTL, for Time To Live. This field will be used to specify the number of peers the message could go through before its erasing of the network. This message is broadcasted to all neighbors the peer owns.
The last and not least class of messages needed for the system concerns search over the network. To proceed search over a P2P network we need at least three messages, one for sending the request (query), one for the answer (queryHit), and one for the demand of download (push).
The query message's payload will be composed of two fields. One of these fields will concern the search criteria, and the other field will concern the nature of the request. This nature is devised into two categories, the block's one, and the data set and task of a component block. Indeed, as we have already spoken about the search mechanism could be used in order to find a part of the data set and task, and to improve robustness by setting on an automatic search for a peer, which is supposed to update a block-component, when another peer detects a long non-updating of this block-component. Furthermore, we have to fix out a quantitative criterion for the search for a part of data set and task, for example n peers to go through or m results or both.
The queryHit message, like query, will be composed of two fields. One for answer, and one for the nature of the request this message is answering to.
The last message is composed of the destination of the message, the source, and the name of the wanted data set or task.
These three messages are carried over the network from peer to peers. But when the destination peer receives a push message, it will send the request by using a direct connection and a protocol that could go through firewall, for example HTTP. We can schematize this scheme as follow: As we have already spoken

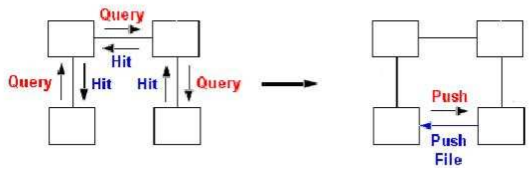

Figure 4: Search Mechanism

about at the beginning of this section, we can build such a system thanks to two kinds of technology, the software's one or the middleware's one.We can notice that in the case of an implementation based upon the software technology we could use a tools box like JXTA project.

6.2 Numerical experiments

In this section we have presented possible implementations of a P2P global computing system, based on the software or middleware technology. We have built a middleware architecture following the above concepts, which leads us to design a system based on the model of figure 4, and we present here first results obtained for an asynchronous auction algorithm applied to the solution of shortest path problem in a network [4]. This algorithm finds all the shortest path from any node in the network to a particular one. The communication of the iterate vector (the price of each node) happens every n iterations (we have picked up n$\bar{4}$000).
We used for this implementation CORBA and the Java language. The system we used has been built thanks to the LAN (100Mbps) architecture of our laboratory. We used several heterogeneous (OS, Processor?) computers in a non-dedicated mode. We tested

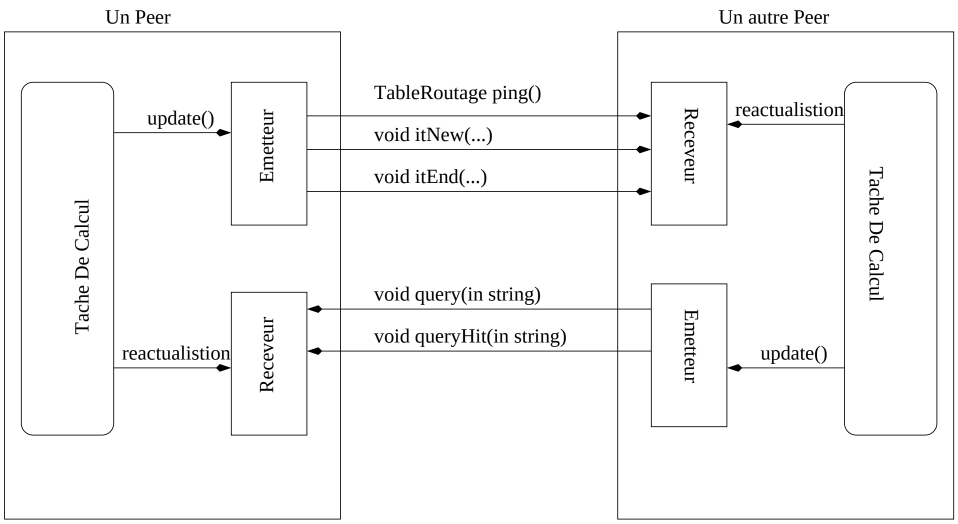

Figure 5: **A first architecture.** Squares represent CORBA objects with an IDL interfaces and circles represent non-CORBA objects. Arrows represent the calling of the remote methods; the orientation of these arrows means for example that the sender calls the itSetup method of the receiver.

the above algorithm, for finding shortest path in a network, for several topologies created by BRITE.
We studied four particular topologies. All of these topologies follow power-law relationship (model BA in BRITE). These networks were built incrementally, and have got a heavy tail node placement.
The constraints, of all the links created in the network, correspond to the delay of the associated links. Table 5 displays the solution time in second for problems with different numbers of nodes, N, numbers of edges, A, and numbers of peers. We note that the performance of asynchronous auction algorithm is good. The lost of efficiency when the number of peer increases, can be explained thanks to three facts. The first one is the utilization of CORBA and Java, which is a factor of slowdown. The second factor of slowdown is the architecture, which is not convenient for this particular algorithm because the time of computation is very small as compare to the communication time (we need almost 20000 iterations per peer and the ratio time of communication over time of computation is nearby 3). And the last factor, for this lack of performance, is the heterogeneous nature of peers and the non-dedicated use of computers.

7 Conclusion

We have presented in this work possible solution to build P2P Global Computing systems. Especially, we show that asynchronous iterative algorithm could be implemented over such a network, thanks to a set of messages. On the other hand synchronous iterative algorithms might be difficultly implemented, because it is supposed that a rendezvous point is established, in order to set a barrier of synchronization.
Concerning the initial repartition of the data set and task, we show that the efficiency of the three algorithms, we have proposed, depends on the network's topology. That's why an efficient control of this topology, via for example ping/pong mechanism, is critical to improve the global efficiency.
The first architecture we propose in this document proves that it is possible to build such an architecture easily, in a future work we plan to study an other implementation, which may be more efficient (C++,

Nombre de Peers	N=1000, A=2000	N=1000, A=4000	N=2000, A=4000	N=2000, A=8000
1	7.53	6.39	14.5	17.2
5	5.7	4.2	9.5	11.5
10	2.45	2.2	4.9	6.1
15	2.3	2.02	4.7	6.0
20	2.0	1.7	4.6	5.8

Table 5: First results of the asynchronous shortest path auction algorithm over a P2P Global Computing System.

omniORB for example).

References

[1] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A.Huberman. Search in power-law networks. *Phys. Rev.*, 2001.

[2] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *J. Assoc. Comput. Mach.*, 2:226–244, 1978.

[3] Mayank Bawa, Hector Garcia-Molina, Aristides Gionis, and Rajeev Motwani. Estimating aggregates on a peer-to-peer networks. Technical report, Stanford University, Database group, 2003.

[4] D. P. Bertsekas. An auction algorithm for shortest paths. *SIAM J. on Optimization*, 1:425–447, 1991.

[5] Dimitri Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical methods.* Prentice-Hall Inc., 1989.

[6] Brian F. Cooper and Hector Garcia-Molina. Sil: Modelling and measuring scalable peer-topeer search networks. Technical report, Stanford University, Database group, 2003.

[7] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. Technical report, Stanford University, Database group, 2001.

[8] D. El Baz. Asynchronous gradient algorithms for a class of convex separable problems. *Computational Optimization and Applications*, 5:187–205, 1996.

[9] Didier El Baz. Contribution à l'algorithmique parallèle. le concept d'asynchronisme : étude théorique, mise en œuvre et application, 1998. Habilitation à diriger des recherches.

[10] Didier El Baz. An efficient termination method for asynchronous iterative algorithms on message passing architectures. In *Proceedings of the International Conference on parallel and distributed Computing Systems*, volume 1, pages 1–7, Dijon 25-27September 1996.

[11] M. Faloutsos, P. Faloutsos, and C Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM '99, Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 251–262. L. Chapin ACM New York, 1999.

[12] A. Frommer and D. Szyld. Asynchronous iterations with flexible communication for linear systems. *à paraître dans Calculateurs Parallèles*, 1998.

[13] Gnutella. specification v 0.4: http://www.stanford.edu/class/cs244b/gnutella_protocol_ 0.4.pdf.

[14] J. C. Miellou. Algorithmes de relaxation chaotique à retard. *RAIRO*, pages 55–82, 1975.