Towards a Peer-To-Peer Platform for High Performance Computing

Nabil Abdennadher*, Regis Boesch**
University of Applied Sciences, Western Switzerland
** abdennad@eig.unige.ch ** rboesch@eig.unige.ch*

Abstract

This paper describes a Global Computing (GC) environment, called XtremWeb-CH (XWCH). XWCH is an improved version of a GC tool called XtremWeb (XW). XWCH tries to enrich XW in order to match P2P concepts: distributed scheduling, distributed communication, development of symmetrical models. Two versions of XWCH were developed. The first, called XWCH-sMs, manages inter-task communications in a centralized way. The second version, called XWCH-p2p, allows a direct communication between "workers". XWCH is evaluated in the case of a real high performance genetic application.

1. Introduction

High Performance Computing (HPC) landscape has radically changed since the end of the last decade. Based initially on the use of parallel and vectorial computers equipped with specific development environments, computing power consumers are adopting a new approach which takes advantage of the Internet development. The idea consists on deploying High Performance applications on anonymous connected computers by using their available resources. Indeed, the challenge today is to extract, at low cost, a reasonable computing power from a widely distributed platform (by executing interactive applications) rather than extracting the maximum power from a local supercomputer (by executing batch applications). In another words, the majority of the world's computing power is no longer in supercomputer centers and institutional machine rooms. Instead, it is now distributed in a hundred of millions of personal computers all over the world. This concept is known as Global Computing (GC).

The majority of GC projects adopted a centralized structure based on a Master/Slave Architecture: SETI@home [1], Entropia [2], United Devices [3], Parabon [4], XtremWeb [5], etc. A natural extension of the GC consists on distributing the "decisional degree" of the master in order to avoid any form of centralization. Thus, architectures such as Clients/Servers and Master/Slaves would be withdrawn. This concept, known as Peer-To-Peer (P2P), was successfully used to share and exchange files between computers connected to Internet. The most known projects are Gnutella [6] and Freenet [7]. Indeed, file sharing is well adapted to this model. However, the use of P2P in the field of HPC raises several theoretical and practical problems. Dynamic scheduling algorithms for parallel/distributed applications can not be easily distributed. P2P Computing also goes against the policies and safety techniques largely used nowadays on Internet: Firewalls, NAT addresses, etc. The objective of these techniques is to protect resources connected to Internet from any voluntary or involuntary abusive use. Internet is then partitioned in several protected zones which are unable to cooperate mutually. Problems related to the development of a true P2P environment for HPC needs remain open.

This document describes a GC environment, called XtremWeb-CH (*XWCH*), which converges towards a P2P system. *XWCH* is an improved version of a GC tool called XtremWeb (*XW*). *XWCH* tries to enrich *XW* in order to match P2P concept: distributed scheduling, distributed communication, development of symmetrical models, etc. In P2P systems, nodes are assumed to be customers and servers at the same time. Although it is utopian, this idea was retained as guide line in the *XWCH* project.

This document is organized as follows: paragraph 2 presents the features that should be satisfied by a GC platform in order to be considered as a real P2P system. Paragraph 3 introduces the *XW* tool in its original version. Paragraph 4 details the new concepts *XWCH* introduces compared to *XW*. Paragraph 5 presents the experiments carried out in order to

evaluate *XWCH*. Lastly, the paragraph 6 gives some perspectives of this research.

2. What is a real Peer-To-Peer system?

A true P2P environment should satisfy four criteria:

- *Natural scalability*: A P2P system should be scalable by itself and not by "doping". For that purpose, the performance of the system should be provided by its distributed structure: distributed algorithms, distributed warehouses, distributed scheduling algorithms, etc. This structure should allow open access and search procedures. The search engine should take into account the dynamic nature of the network. The system should be based on a demand-driven computation model: users' queries are only processed when needed and prior results are stored in warehouses, where they can be accessed later on.
- *Symmetric view*: a node belonging to a P2P platform should be server and client at the same time.
- *Platform heterogeneity*: The system should support heterogeneous architectures (hardware) and platforms (software and operating systems). Since these resources are anonymous, the system should take into account all administration policies implemented by local administrators.
- *Multi-service*: The system should be able to serve any kind of request: HPC, file sharing, etc. We believe that we cannot design a system that can satisfy every user's needs. However, the system should be able to supply users with adequate tools that allow the implementation of specific services not initially foreseen.

Systems like *Gnutella* and *Freenet* satisfy the three first criteria, but these systems are mono-service since they only target file sharing needs. *XtremWeb*, *Seti@home*, *Entropia* and other GC environments do not satisfy any of these criteria. They are based on a non symmetric view (Master/Slaves) and exclusively HPC oriented. They are not scalable since the master is overloaded when the number of slaves increases. The only tool which seems to satisfy all these constraints is WOS (Web Operating System) [8]. Unfortunately, this tool remained in a purely conceptual state and no prototype was born.

3. XtremWeb

XW is a GC research project carried out at Université d'Orsay (France). Like other Large Scale Distributed Systems (LSDS), *XW* platform uses remote resources (pocket computers, PCs, workstations, servers) connected to Internet to execute a specific application (client). The aim of *XW* is to investigate how a LSDS can be turned into a High Performance Parallel Computer. *XW* belongs to the more general context of Grid research and follows the standardisation effort towards Grid Services [9]. *XW* satisfies the three main constraints imposed by any Large Scale Distributed Environment: volatility, heterogeneity and security.

Security is particularly difficult in the context of LSDS because it's impossible to trust hundreds of thousands resources. Three main security problems are linked to GC and P2P systems:

- Data integrity/privacy: This problem could be resolved by applying the well known solutions of encryption, public/private keys, etc.
- Protection of participating resources: No aggressive application should be able to corrupt data or system of any participating resource. Sandboxing is the well known technique to resolve this problem. The idea consists on filtering the system calls which appear to be the main security holes of recent operating systems. [10] explains how does *XW* use the sandboxing to resolve the resource protection problem.
- Result certification procedure: This problem is linked to the lack of trust regarding the result provided by the remote resource. Indeed, there is no way to control precisely what happens on a participating resource. Faulty and malicious behaviour must be detected.

A typical *XW* platform is composed of one coordinator and several workers (remote resources). The coordinator is a three-tier layer allowing connection between clients and workers through a coordination service. This layer is designed so as it allows the mobility of clients and the volatility of workers.

3.1 The coordinator

The coordinator is a three-tier architecture which adds a middle tier between client and workers. There is no task direct submission/result transfer between clients and workers. The coordinator accepts task requests coming from several clients, distributes the tasks to the workers according to a scheduling policy, transfers application code to workers if necessary, supervises task execution on workers, detect worker crash/disconnection, re-launches crashed tasks on any

other available worker, collects and store task results to client upon request.

The coordinator is composed of three services: the repository, the scheduler and the result server. The repository is an advertisement services. It publishes services (client applications) to make them available through standard communication ports (Java RMI, XML-RPC). These applications/services are first read from a database and inserted into the task set. The scheduler is the service factory. It instantiates applications and manages their life cycle. It starts them on workers (a task is an instantiation of service or application), stops them as expected and corrects faults (if any) by finding available workers to re-launch them. Finally the result server collects results as they are provided by workers.

3.2 Workers

The worker architecture includes four components: the task pool, the execution thread, the communication manager and the activity monitor. The activity monitor controls whether some computations could take place in the hosting machine regarding some parameters determined by the worker configuration (% CPU idle, mouse/keyboard activity, etc.). The tasks pool (worker central point) is managed by a producer/consumer protocol between the communication manager and the execution thread. Each task should be in one of the three states: *ready* to be computed, *running* or *saving*. The first state concerns downloaded tasks, correctly inserted into the pool. The second state is for tasks being computed. The last state corresponds to tasks which need to upload result file to the result server. The communication manager ensures communication with the coordinator; it downloads task files (binaries and input data) and upload results, if any. When download completes, the task is inserted into the task pool. The execution thread extracts the first available task from the pool, recreates the task environment as provided by the client (binary code, input data, directories structure, etc.), writes on disk the task status, starts computation and waits for the task to complete. When the task completes, it creates the results file which includes standard output and updates task status on disk. The execution thread finally marks the task state as completed, allowing the communication manager to send results. It then expects notification from the result server to send again in case the upload went wrong or definitively remove the task.

In its original version, *XW* applications are standalone modules. The system does not support any interaction between different modules. However, developers can use asynchronous Remote Process Call called *XWRPC* in order to distribute (parallelize) their applications [11].

4. XtremWeb-CH

XtremWeb-CH (*XWCH*) is an upgraded version of *XW*. The aim of *XWCH* is to build an effective Peer-To-Peer LSDS which satisfies the four criteria detailed in paragraph 2. *XWCH* adds four functionalities to *XW*:

1. Automatic execution of Parallel and Distributed Applications (PDA)
2. Automatic detection of the smallest granularity that can be implemented according to the number of available workers.
3. Support of direct communication between workers.
4. *XWCH* provides a set of monitoring tools allowing users to visualize the execution of their applications.

4.1 Automatic execution of Parallel and Distributed Applications (PDA)

In *XW*, jobs submitted to the system are standalone. In case of PDA, communicating modules are executed as separate jobs (tasks). It's the user responsibility to link manually output and input data of two communicating tasks. Contrary to this approach, *XWCH* supports the execution of a whole PDA. A PDA is a set of communicating modules that can be represented by a data flow graph where nodes are modules and edges are communications inter-modules (Figure 1). According to the semantics of the PDA, modules can have the same or different codes. In figure 1, modules having the same shape have the same code.

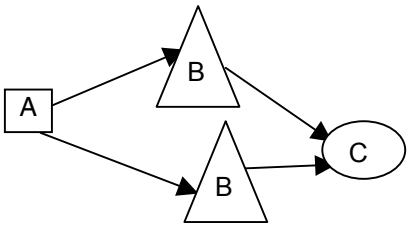

Figure 1. Data flow graph representing a PDA application

The data flow graph is represented by an XML file whose syntax is detailed in Figure 2.

An application is composed of several modules (*Module* element in Figure 2). A module is represented by a source code and can have several binary versions (*Binary* element in Figure 2). A task is an instantiation of one module. Thus, several tasks can correspond to

the same module. The maximum number of tasks for a given module is fixed by the *Restriction* element. This element fixes the smallest granularity of the application during its execution. It can be extracted from the "state" of the platform just before the execution time (see paragraph 4.2 for details). It represents the maximum number of workers that can be used to execute the corresponding module.

Precedence rules between tasks are described by *Task* elements. A task can have several inputs (*Input* element in Figure 2) but only one output (*Output* element in Figure 2). The element *cmdLine* indicates arguments/parameters used by the task. This field is optional.

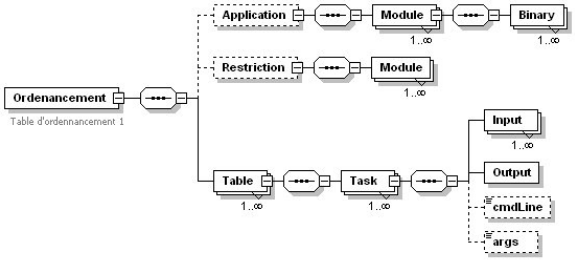

Figure 2. XML syntax of a PDA application

A PDA is thus, represented by:

- its XML file representing its data flow graph,
- the binary codes of its modules. Let's recall that one module can have several binary codes,
- its input data.

These files are compressed into one file.

XWCH can be perceived as a layer on *XW* that takes into account the communications between tasks belonging to the same PDA. In this context, a task belonging to a given PDA is considered by *XW* as a standalone application.

A client can submit his PDA to *XWCH* by uploading its corresponding compressed file. In addition to the three states *ready*, *running* and *saving*, *XWCH* adds a fourth state: *blocked*. Tasks of a given PDA are initially *blocked* and cannot be assigned to any worker, since their input data are not available. Only tasks whose input data are given by the user are in *ready* state and can be allocated to workers. When they are assigned to a worker, they move from *ready* to *running* state. Input data needed by *blocked* tasks are progressively provided by *running* tasks which finish their processing. *XWCH* detects the *blocked* tasks which can pass to ready state and can, thus, be assigned by the scheduler to a worker.

4.2 Granularity of the PDA

In parallel computing, the selected size of the grain (granularity) depends on the application and the number of processors in the target parallel machine. This number is generally known and fixed during the execution. Thus, the granularity is fixed during the development of the application. In our context, the computer is the network, workers are free to join and/or leave the GC platform whenever they want. The exact number of available workers is known just before the execution and could be varied during the execution. As a consequence, the granularity should be fixed only at execution time. The client indicates, in the XML file describing his PDA (*Restriction* element), the number of workers each module of the PDA can use during its execution (*max_workers*). This number depends on the semantic of the application and should be provided by the client. To deploy an application on *XWCH*, three steps are required:

1. Discovery step: Search for a set of available workers to execute the PDA. The number of workers should be less or equal to *max_workers*.
2. XML generation step: this step consists on generating the XML file of the application to be deployed according to the number of available workers. In general, it's the user responsibility to generate this file. However, for a specific family of applications, this file can be automatically generated according to the *XWCH* platform status: number of available workers, network status, etc. Thus, the number of tasks is fixed just before the execution. In another words, granularity of the parallelization is dynamically fixed according to the number of available workers and the state of the targeted P2P platform.
3. Execution step: the application is launched on the *XWCH* platform.

4.3 Direct communication

Two versions of *XWCH* were developed. The first, called *XWCH-sMs*, manages inter-tasks communications in a centralized way. The second version, called *XWCH-p2p*, allows a direct communication between workers without passing by the coordinator

In the *XWCH-sMs* (slave-Master-slave) version, workers cannot directly communicate, they cannot "see" each other. Any communications between tasks take place through the coordinator. This architecture overloads the coordinator and could affect the application performances.

In order to cure the gaps of the *XWCH-sMs* version, it is necessary to have direct worker-to-worker communications. In other term, the worker executing module *A* (called *worker A* in Figure 3) must be able to directly send its results to *workers B* and *C*.

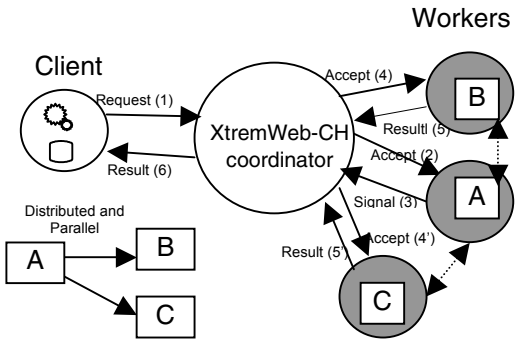

Figure 3. Execution of a PDA on a *XWCH-p2p* platform

The *XWCH* coordinator can, thus, allocate tasks *B* and *C* to two available workers. Every worker receives the binary code of the module it will execute and the necessary information relating to its input file (IP address, path and name of the input file). Data transfer between workers *A* and *B* (*resp.* C) can thus take place on the initiative of the receiver. This version called *XWCH-p2p* has two main advantages:

1. it discharges the coordinator from data routing and,
2. it avoids the duplication of communications.

In this context, the coordinator keeps only the task scheduling management. *XWCH-p2p* tends towards the Peer-To-Peer concept which one of its principles is to avoid any centralized control.

Direct communication can only take place when the workers can "see" each other. Otherwise (one of the two workers is protected by a firewall or by a NAT address), direct communication is impossible. In this case, it is necessary to pass by an intermediary (*XWCH* coordinator for example). This scenario is similar to *XWCH-sMs* version. However, to avoid overloading the coordinator, one possible solution consists on installing a relay machine, called "data collector" which acts as an intermediary. This machine is used by worker *A* (in our example) to store its results and by workers *B* and *C* to seek their data. "Data collector" machine is chosen by the user when launching the application. This machine must be reachable by all workers contributing to the execution of the concerned application.

4.4 Monitoring tools

XWCH proposes a package of tools allowing the user to debug and/or visualize the progress of his PDA execution:

- Tasks allocation: The user can "spy" the execution of his PDA. He can follow the allocation of tasks (which worker is executing which task)
- Progress of tasks execution: When executing, every task can send progress report to its worker informing it about its state. Currently, this progress report is expressed in term of percentage of execution. 60% means that the task has finished 60% of its execution.
- Step by step execution: It's a debugging mode. When activated, every task sends messages to the worker. These messages are inserted in the source code by the developer.

5. Experimental measures

The purpose of this section is to assess the performances of *XWCH* in a real case of a CPU time consuming application. *XWCH* was evaluated in the case of a phylogenetic application. Phylogenetic is the science which deals with the relationships that could exist between living organisms, it reconstructs the pattern of events that have led to "the distribution and diversity of life". These relationships are extracted from the Desoxyribo Nucleic Acid (DNA) sequences of species. A phylogenetic tree, also called life tree, is then built to show relationship among species. This tree shows the chronological succession of new species (and/or new characters) appearances.

In a medical context, the generation of a life tree for a family of microbes is particularly useful to trace the changes accumulated in their genomes. These changes are due, inter-alia, to the "reaction" of the virus to the treatments (antibiotic for example).

A multitude of applications aiming at building phylogenetic trees are used by the scientific community. These applications are known to be CPU time consuming, their complexity is exponential (*NP-difficult* problem). Approximate and heuristic methods do not solve the problem since their complexity remains polynomial with an order greater than 5: $O(n^m)$ with $m > 5$. Parallelisation of these methods could be useful in order to reduce the response time of these applications.

The *Tree Puzzle* method [12] [13] is one of the heuristic techniques used for the generation of phylogenetic trees. [14] and [15] propose a parallel implementation of the Tree Puzzle method written in C

and using Message Passing Interface (MPI) communication routines. This implementation, particularly optimized for a cluster of computers, was adapted to our *XWCH* platform. MPI routines were replaced by file transfers. However, no code optimization was done. Our goal is not to develop an optimized version of the *Tree Puzzle* algorithm for an *XWCH* platform, but to validate choices retained within the framework of the *XWCH* project.

The input data of *Tree Puzzle* algorithm is represented by the DNA sequences of the species to be classified. A DNA sequence is modeled by a chain of few hundreds of characters. The algorithm generates a structure representing the phylogenetic tree of the species given in the entry. The *Tree Puzzle* algorithm is not detailed in this document. However, its structure, expressed in term of tasks (data flow graph) is given in Figure 4. This structure is common to several families of PDA. An XML file generator was developed. The goal is to automatically generate this file according to the number of available workers and structure of exchanged data between tasks.

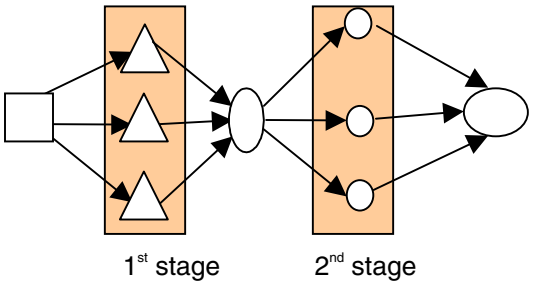

Figure 4. Structure of the *Tree Puzzle* algorithm

Tasks belonging to the 1st stage (*resp.* 2nd stage) have the same code, their number is equal to $N - 3$ where N is the number of DNA sequences. The number of tasks belonging to the 2nd stage is variable and can be chosen by the programmer, but can never exceed N.

Tree Puzzle application was executed on *XWCH* (*XWCH-p2p* and *XWCH-sMs* versions) with two jets of input data: 64 (128 tasks) and 128 sequences of DNA (256 tasks). *XWCH* was installed on more than 100 heterogeneous PC (Pentium 2, 3, 4) with Windows and Linux operating systems distributed between two sites: University of Applied Sciences (Geneva-Switzerland) and Polytechnic School of Lille (France).

The 2nd stage of the application consumes 70% of the processing time. For this reason, tests focused on varying the number of tasks at this stage. In figure 5 (*resp.* 6), *Tree Puzzle* application was executed by varying the number of tasks of the 2nd stage: 8, 16, 32 and 64 (*resp.* 32, 64 and 128). When the number of DNA sequences is equal to 128 (Figure 6), and with a number of sequences equal to 8 (*resp.* 16), the execution time is estimated to 7 days (*resp.* 3 days).

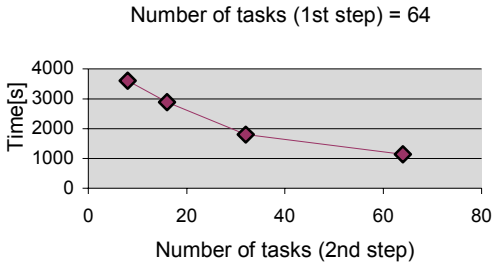

Figure 5. Execution time of *tree puzzle* algorithm. Number of sequences = 64

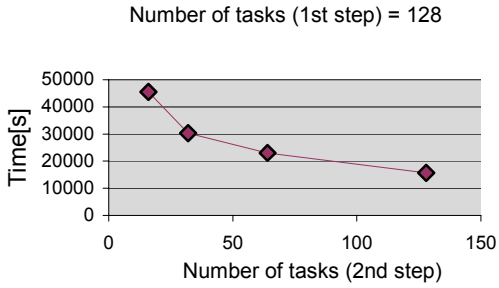

Figure 6. Execution time of *tree puzzle* algorithm. Number of sequences = 128

During the execution, some of the available workers were not "exploited" by the application. Indeed, the number of tasks in the 2nd stage never exceeds that of workers. On the other hand, during the execution of the 1st stage, task allocation process is faster than the execution itself. Consequently, the scheduler assigns 1st stage tasks to workers having already executed the same code (workers already having the binary code).

The objective of these measurements was to validate our approach. In this context, no optimization was brought to the parallel *Tree Puzzle* algorithm. However, several improvements could be carried out in order to adapt the algorithm to the targeted platform. Indeed, a specific parallelization of the *Tree Puzzle* algorithm adapted to *XWCH* platform could decrease the response time of the application.

Figure 7 shows execution times of another parallel/distributed application (parallel *mergesort* algorithm) when executed on *XWCH-sMs* and *XWCH-p2p versions*. Indeed, communication costs of this

application are more important than those generated by the Tree Puzzle algorithm.

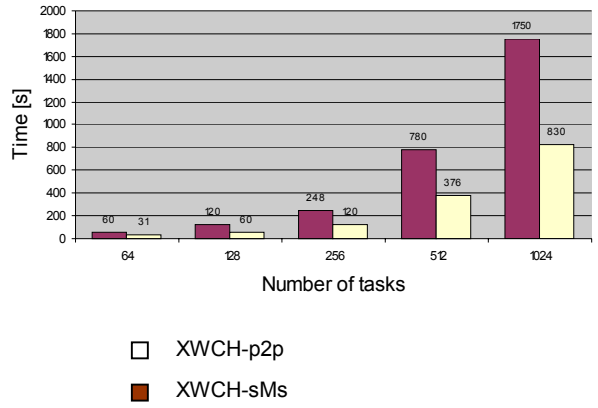

Figure 7. *XWCH-sMs* vs. *XWCH-p2p*

These Measurements (Figure 7) are foreseeable: *XWCH-sMs* version consumes twice more communications than the *XWCH-p2p* version.

Figure 8 shows the output traffic (expressed in term of bits) generated by *XWCH* coordinator during 4 hours and 30 minutes. During this period, two applications were executed, they correspond to the two peaks of figure 8 (phases II and V). A traffic analyser was used to obtain these measurements. The x-axis (x) represents time while the co-ordinates (y) represent number of bits generated by the coordinator.

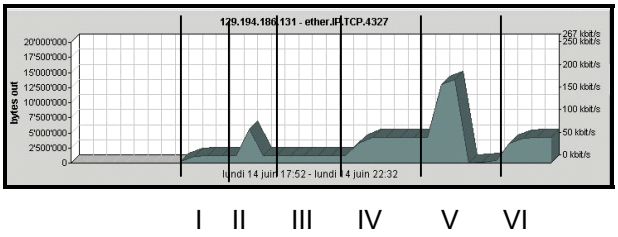

Figure 8. Output traffic generated by *XWCH* coordinator

Phase I corresponds to the launching of 5 workers. The traffic generated by the coordinator corresponds to the replies it generates following the "work request" calls sent by the workers. During this phase, no application is deployed. Phase II corresponds to the launching of a PDA: a sort application based on *mergesort* algorithm. This traffic corresponds to data and binary codes transmitted by coordinator towards workers. Phase III is similar to phase I, it corresponds to the "work request" calls sent by the 5 workers after they end their execution. Phase IV corresponds to the launching of 24 workers.

Phase V corresponds to the execution of the same application with a larger size of input data. The peak at the beginning of this phase shows the transmission of binary codes and data from coordinator to workers. When workers execute tasks, the coordinator outgoing traffic is null. Indeed, communications take place directly between the workers. This phenomenon does not appear in the first execution because of the short execution time of the application. Phase VI is similar to phase IV.

These measurements show that the average output traffic generated by the coordinator is equal to 3.24 kbits/s by worker.

6. Conclusion

This paper presents a new GC environment (*XtremWeb-CH*), used for the execution of high performance applications on a highly heterogeneous distributed environment. *XWCH* can support direct communications between workers, without passing by the coordinator. The execution of a testbed application (generation of phylogenetic trees) has demonstrated the feasibility of our solution. Other experiments are in progress to evaluate *XWCH* in other High Performance applications cases.

One of the ideas that could constitute the perspectives of this work is to extend the *XWCH-p2p* version in order to converge towards a true P2P system which one of its principles is to eliminate any centralized control. The current version of XWCH allows the decentralization of communications between workers. The next step consists on designing a distributed scheduler, executed by workers. This scheduler should avoid allocating communicating tasks to workers that can not reach each other. Although not specifically discussed, this approach offers a strong basis onto which we could develop distributed and dynamic scheduler and should confirm and reinforce the tendency detailed in section 2.

7. References

[1]: http://setiathome.berkeley.edu/

[2]: http://www.entropia.com/

[3]: http://www.ud.com/home.htm

[4]: http://www.parabon.com/

[5]: Gilles Fedak et al. *XtremWeb : A Generic Global Computing System*. CCGRID2001, workshop on Global Computing on Personal Devices. Brisbane, Australia. May 2001. http://xtremweb.net

[6]: KAN G., *Peer-to-Peer: harnessing the power of disruptive technologies*, Chapter Gnutella, O'Reilly, Mars 2001.
[7]: Ian Clarke. *A Distributed Decentralised Information Storage and Retrieval System*. Division of Informatics. Univ. of Edinburgh. 1999. http://freenet.sourceforge.net/

[8]: Babin, G; P. Kropf; and H. Unger. *A two-level communication protocol for a Web Operating System: WOS*. Vasteras, Sweden, Aug 1998. In IEEE Euromicro Workshop on Network Computing, 939–944.

[9]: I. Foster, C. Kesselman, J. Nick, and S. Tuecke. *Grid Services for Distributed System Integration.* IEEE Computer, pages 37-46, June 2002.

[10]: Franck Cappello et al. *Computing on Large Scale Distributed Systems: XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid.* In Future Generation Computer Science (FGCS), 2004.
[11]: Samir Djilali. *P2P-RPC: Programming Scientific Applications on Peer-to-Peer Systems with Remote Procedure Call.* GP2PC2003 colocated with IEEE/ACM CCGRID2003. Tokyo Japan, May 2003.

[12]:http://biowulf.nih.gov/apps/puzzle/tree-puzzle-doc.html

[13]: http://www.tree-puzzle.de/

[14]: http://www.dkfz.de/tbi/tree-puzzle/

[15]: *Heiko A. Schmidt*, Phylogenetic Trees from Large Datasets, 'Ph.D.' in Computer Science, Düsseldorf, Germany, 2003.