XtremWeb: Building an Experimental Platform for Global Computing

Cécile Germain, Vincent Néri, Gilles Fedak, and Franck Cappello

Laboratoire de Recherche en Informatique.
Université Paris Sud
http://www.XtremWeb.net

Abstract. Global Computing achieves highly distributed computations by harvesting a very large number of unused computing resources connected to the Internet. Although the basic techniques for Global Computing are well understood, several issues remain unadressed, such as the ability to run a large variety of applications, economical models for resource management, performance models accounting for WAN and machine components, and finally new parallel algorithms based on true massive parallelism, with very limited, if any, communication capability. The main purpose of XtremWeb is to build a platform to explore the potential of Global Computing. This paper presents the design decisions of the first implementation of XtremWeb. We also present some early performance measurement, mostly to highlight that even some basic performance features are not well understood yet.

1 Introduction

The XtremWeb project is dedicated to the study of a particular execution model in the general framework of Global (or Meta) Computing. In this model, all the computing power is provided by volunteer computers. These computers offer some of their time to execute a piece of a very large application, under more or less severe restrictions on the use of this time.

The XtremWeb environment and the other related ones like Entropia.com, Distributed.net, Seti@home [3] are Web extensions of cycle stealing concepts originally intended for networks of workstations. Condor [6], Globus [10], Atlas [5], Nimrod/G [1] and some other systems have addressed the issues of cycle stealing in the context of LAN environments. The main characteristics of a LAN and a Web environment drastically diverge.

- *Number of connected machines*: hundreds for a LAN; millions for the Web.
- *Security and protection*: stations inside a LAN are well identified; machines on the Web could be very malicious.
- *Stability* : the Mean Time Between Failure (MTBF) on a LAN is several days, versus several minutes on the Web. Moreover, the LAN stations are permanently attached to the network and belong to the institutions that may steal their cycles, while the machines on the Web do not belong to the project that wants to borrow them. In particular, the owner of a machine may unplug its machine, either physically, or from the project at any time.

These differences make cycle stealing on Internet much more challenging than the simple adaptation of existing systems to the Web. However, cycle stealing on the web may provide unprecedented computing and storage power. It may also provide a new framework for the study of parallel algorithms, mostly because an application may use a nearly unlimited number of processors poorly coupled.

The XtremWeb project aims at building a platform for experimenting with global computing capabilities. The Global computing platform is designed to be a substrate for plug-in experiments. Some issues to be addressed are: sizing of the environment components (servers, network, workers) according to applications features; high performance and secure execution (relies on program isolation); modeling resource and workload management as inputs for scheduling algorithms; and the impact of the application characteristics, either compute- or data-intensive.

The next section presents main issues of global computing systems. The XtremWeb architecture is detailed in section 3. The first implementation is presented in section 4. Section 5 displays some early experiments. The next section exemplifies a typical application. The last section concludes.

2 Global Computing Issues

All Global Computing systems must exhibit a set of desirable properties. We quote them and discuss how they specialize in the XtremWeb case.

- *Scalability.* It must scale to hundreds of thousands nodes, with corresponding performance improvement. The target performance is throughput, not latency of individual computations.
- *Heterogeneity.* Target machines are personal computers and workstations. Load-sharing facilities (LFS, Condor), or batch systems, such as the IBM Load-Leveler are not considered as a part of the chain of contribution: the workstations will decide on an individual basis if and when they are willing to contribute.
- *Availability.* The owner of a computing resource must be able to define a policy which limits the contribution of the resource.
  The policy is defined by a type of activity at the workstation level, and is binary: when this type of activity makes a transition from "off" to "on", the workstation is immediately released, whatever loss in the global computation this may imply.
- *Fault tolerance.* True hardware or software faults, including unplugged laptops, and unexpected computation aborts due to the availability policy, must be managed identically, as an interrupted computation does not have the right to use local computer resource to save any of its state or to signal the event to the global system.
- *Security.* All participating computers should be protected against malicious or erroneous manipulations, and the global computation result should not be exposed to be tampered with.
- *Usability.* The system should be easy to deploy and to use.

3 The XtremWeb Design

3.1 Application Scope

XtremWeb focuses on an important class of applications : the *embarrassingly parallel* ones, also coined as *multi-parameter*. These applications consist of a large number of instances of the same computation applied to varying input parameters. In this case, each computation completes independently of the others, and the information flows only between the worker and the dispatcher. If one computation fails, because the worker has been preempted, other ongoing computations will not be affected.

3.2 The Pull and Steal Model

The XtremWeb execution model combines a pull model and a cycle-stealing scheme. In the pull model, workstations (*workers*) withdraw work from a central agent (*the dispatcher*), in opposite to a push model, where workstations are borrowed by an external agent. The cycle-stealing scheme is characterized by constraints that can prevent the computation to complete, even without any computer or network failure.

The paradigm of the pull-and-steal model is the screen-saver scheme, as exemplified by the popular SETI@home project [3] and Nimrod/G [1]. When a participating workstation is not interactively used, as detected by a screen saver utility, the workstation participates in the global computation. As soon as the user comes back to the workstation, the screen saver vanishes and so does the ongoing computation; all unfinished work is lost.

The pull model is not limited to the screen saver scheme, and not even to cycle-stealing. It can be extended to a strategy for dynamic load balancing on a large set of workstations, each of them with a variable level of commitment to the solution of the global application. For instance, some of them may be willing to contribute only if their activity is not above a certain, locally determined, threshold, while other ones may be fully devoted to the application. One of the objectives of the XtremWeb architecture is to accommodate these various contributing policies, including the screen-saver one, in an unified framework.

3.3 One-sided Communication

In the framework of Massively Parallel Processing, one-sided communication has been exemplified by various implementation of *get* and *put* primitives. The main idea is that one participant can perform all information transfer, either put, (i.e. writing to a remote partner), or get, (i.e. reading from a remote partner). The cooperation of the accessed remote partner is not required at the programming level, even if an underlying infrastructure must ensure the actual access. This contrasts with message-passing schemes, where both partners must collaborate through paired send/receive calls. In the distributed computing framework, one-sided communication is provided by RPC (Remote Procedure Call) or RMI (Remote Method Invocation), following the programming model.

All XtremWeb information transfers are controlled by the workers. They perform RMI calls to the dispatcher, and no provision is made for the contrary. The first motivation for this choice is security: with one-sided communication, the workers security is guaranteed by server authentication and protection of the data transmission from the server.

Another motivation is ease of deployment. The security policy of the dispatcher is configurable, while the one of the worker is not. Callbacks from the dispatcher to a worker depend on the last one, and can thus be blocked by firewalls, or require the adoption of very slow protocols such as http.

With this scheme, the dispatcher performance becomes even more critical. While one part of the communication overhead is distributed across the workers, all the control cost is centralized on the dispatcher. The abstract dispatcher must then be instantiated in as many actual dispatchers as necessary to sustain the throughput required by the expected number of workers.

3.4 Native Code Execution

XtremWeb targets high performance. Thus, although the workers protection suggests execution in a virtual environment, typically sand-boxed Java bytecode, performance dictates that the end-user code should remain native.

Like most of the other Global Computing system XtremWeb uses native code execution. However, in contrary to them, XtremWeb allows any workers to execute different and downloadable applications.

New applications are made downloadable following a verification process that is more complicated than the byte code verification of Java virtual machines but less secure. First, only trusted institutions can propose codes to integrate in XtremWeb. Second, the code is executed on dedicated workers. Third the code is encrypted before downloading to workers. Fourth, the code download procedure uses a private-public key to secure the transaction. This verification process cannot prevent from any fault case because testing (second phase) may not execute all code sections with all possible parameters sets. So there is still a risk of execution error, which is not the case, in principle, with bytecodes and virtual machines.

A more flexible way to allow downloadable high performance native code execution is the technique known as Software Fault Isolation [4]. This kind of approach is necessary to allow the execution of any application without deeply checking the application before execution. We plan to evaluate this approach in future version of XtremWeb.

4 Implementation

4.1 Java Based Coordination and Coupling

The first implementation of the XtremWeb infrastructure is completely written in Java. The Java language and APIs provide portability (related to the ease of deployment issue). It also provides language-level constructs for concurrency through the Java Threads, and parallelism through Java RMI. Integrating binary high performance code is straightforward through Java Native Interface. Finally, special-purpose APIs are available for nearly any special functionality required, in particular authentication and encryption through the Secure Socket Layer (SSL) system and vendor-neutral database access through Java DataBase Connectivity (JDBC).

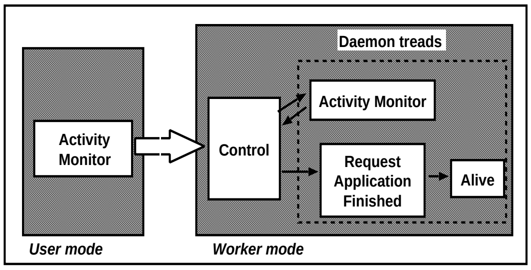

Fig. 1. The worker

Figure 1 shows the worker architecture. In user state, a background process running at low priority monitors the computer activity, following the availability policy, and also the CPU activity for performance prediction service (see below). When the computer becomes available, a new process is launched. This process starts with a control thread, that creates a monitoring thread and a compute thread, and waits forever for the monitoring thread to terminate. The compute thread invokes *WorkRequest* and *getWork*. This calls register the worker to the dispatcher and returns a description of the work to be done as well as the needed work input. Then the compute thread runs the actual computation and invokes *WorkFinished* and *WorkResult* for transferring back the results to the dispatcher. It also launches a thread that periodically invokes *WorkAlive* to signal its activity to the dispatcher. The dispatcher continuously monitors these calls. When a worker has not called for a sufficient long time, the worker computation is considered lost and rescheduled for another worker.

When the monitoring thread detects an increase of the machine load or an external device activation, it terminates immediately, causing the other threads to die. The compute and alive threads run as Java daemon threads. This implementation ensures that whatever synchronization is implied in the invocation of remote methods on the dispatcher, threads cannot become deadlocked.

5 Early Experiments on Server Throughput

One of the main parameter of the performance for XtremWeb application is the ability of the server to answer to work requests. Most transactions between the server and the workers are implemented in terms of Java RMI. RMI overhead for an empty call is more than 500 μs on a 200MHz Pentium, and increases with the complexity of the objects passed back and forth [11]. This high overhead comes from the underlying TCP protocol, and from the design of the serialization procedure, which allows for dynamic class loading. However, the main performance concern for XtremWeb is throughput, not latency since workers are supposed to be spread across a very large geographic area.

XtremWeb throughput has two components: the dispatcher throughput and the aggregate workers throughput. The first one is related to the dispatcher capability for concurrently handling multiple RMIs, while the second one is related to the scheduling policy. Early experiments deal with dispatcher throughput.

The XtremWeb architecture requires a large number of short RMI, corresponding to *WorkAlive* calls. The required RMI throughput is the product of the number of workstations controlled by the RMI call frequency. Predicting the actual RMI call frequency when WAN congestion is taken into account will be the subject of further research. In this section, we report three experiments which measure the dispatcher capacity in terms of RMI throughput. All experiments were conducted on idle machines. The Solaris machine is an UltraSparcII biprocessor at 400MHz, running Solaris 2.7 and Solaris_JDK1.2.1; the Linux machine is a Pentium III biprocessor at 500MHz, running Linux 2.2.13 and jdk 1.2.2 from Blackdown.

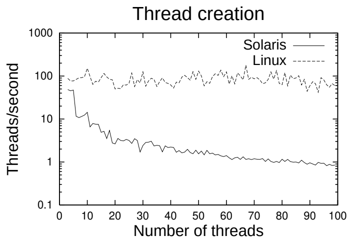

Fig. 2. Threads creation

Each remote method invocation from different machines creates a Java thread. Thus a first concern is the thread creation performance. Fig. 2 shows the result of a simple experiment: a main thread creates a given number of threads, which are affected with Java MIN_PRIORITY before being actually launched. The thread creation rate is not sensitive to the number of running threads in the Sun JDK1.2.1 JVM running on Solaris. In the Linux configuration, the rate rapidly decreases and falls below one thread per second at high load.

The second experiment (fig. 3-A) was conducted so as to isolate the impact of network congestion from RMI calls. A client performs a light RMI call on a server which concurrently runs a fixed number of MIN_PRIORITY Java threads (in practice, the client iterates over RMI calls to measure an average; Java RMI is synchronous, so iterating over RMI does not create any network congestion and averaging makes sense). The remote method is light in the sense that is has no parameters and does not return a value, and only increments a counter. The behavior of the Solaris configuration is what can be expected: the average RMI latency is around 1ms and does not change with the number of adversary threads. The Linux system latency linearly increases with the number of adversary threads. The RMI latencies in presence of only one adversary thread are equivalent in both cases, showing that the RMI implementations are comparable.

Independent experiments have shown that rapid performance degradation with the number of active threads is shared by other JVM implementations on Linux, in particular the IBM one. The reason may be that Java threads are directly mapped to the native Linux threads (one-to-one scheme), which share the process scheduler, and thus are scanned each time the scheduler computes its goodness measures for electing the next process to run. However, the steep curve of fig. 3-A for a low number of adversary threads must reflect a problem specifically associated with thread creation: the actual thread work is so short that is very unlikely that it could invoke the kernel function *schedule ()* while running.

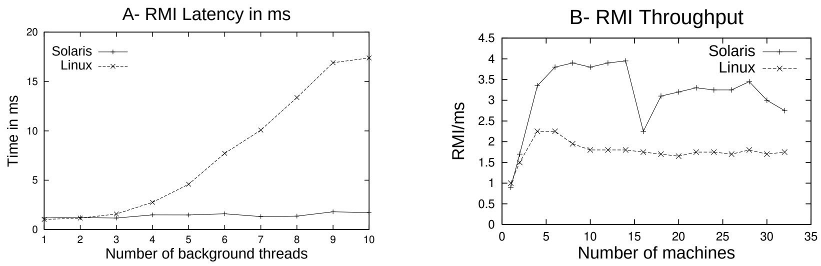

Fig. 3. RMI Latency (ms) in presence of adversary theads and RMI Troughput in RMI/sec

The last experiment (fig. 3-B) simulates at a small scale what could be the behavior of a complete XtremWeb system. Up to 32 heterogeneous machines connected through a LAN perform a light RMI call (averaged as before). The actual RMI throughput is measured. Before entering the RMI loop, the clients are synchronized, so as to ensure approximate simultaneity of the requests. Although Solaris neatly overperforms Linux, the difference is much less pronounced than in the two previous experiments. Two effects are at work here. First, the limited number of machines, possible network congestion and RMI overhead, all limit the number of outstanding requests for thread creation to a much lower level than in the first experiment, where only the thread creation rate was measured. The second effect is that the invoked method is very short. Thus, even if the thread associated with the RMI may wait before creation, not many threads will run concurrently, contrary to the second experiment.

The main conclusions of these early experiment are that server throughput depends on 1) the Java virtual machine implementation (which in turn depends on OS) and 2) the performance of all XtremWeb components : the servers, the workers and the network.

So, we must design a complete methodology (benchmark, experimental platform, result storage and interpretation) in order to measure and understand the respective contribution of each component to the global server throughput.

6 Application Example: The Auger Experiment

6.1 Background

The Pierre Auger Observatory [7] project is an international effort to study the highest energy cosmic rays, above 10^{19} eV. The origin of the very high energy cosmic rays is completely unknown. In fact, until the fortuitous detection of two events above 10^{20} eV, the theory did not allow for them to happen.

Such events are extremely rare: above the energy of 10^{18} eV, only When a cosmic ray particles (primaries) strike the earth's atmosphere, collisions with air molecules initiate cascades of secondary particles, called *air showers*. Two giant detector arrays, each covering 3000 km^2, will be constructed to measure the arrival direction, energy, and mass composition of cosmic ray air showers above 10^{19} eV over many years.

Air showers must also be numerically simulated, in particular by the Aires program.

The simulated results will be compared against the actual observations to infer the physical characteristics (speed, etc.) of the primaries, during the experiments. The inputs of the numerical simulation are the physical parameters of one primary particle plus parameters related to simulation control. The output is the simulated shower particles arriving at the earth level. The number of independent simulations to be run is very large: the simulation is based on a Monte-Carlo scheme, requiring many runs with the same input parameters to compute averages; primaries with various structural and kinetic properties must be simulated; finally, multiple physical models must be simulated. The requirement in computing power is equivalent to 10^6 years of a 300MHz PC per year. At this step of our work and of the Auger experiment, the XtremWeb project is a tentative resource complementary to the production of the classical high-performance computing facilities.

6.2 Implementation of AIRES on Top of XtremWeb

Aires provides an excellent testbed for experiments. The execution time can be predicted with reasonable accuracy from the input parameters. Moreover, with some modifications, the code can be considered as recursive: the shower particles can in turn be considered as primaries for smaller showers. Thus, the granularity can be arbitrarily down-sized [9].

For the Aires simulation, *WorkRequest* and *getWork* are merged, because there are only a few input parameters. Also the decoupling *WorkFinished* and *WorkResult* calls is mandatory. With a typical 10MB result file, the time scale and disk requirement of *WorkResult* is not consistent with the one of the other transactions.

Although the first version will include only a crude scheduler of complete showers based on the time of day, we plan to experiment in particular on the

Rosenberg model [12]. This model considers a fixed startup cost accounting for network latency and a configurable workload, which is exactly the case of Aires.

XtremWeb and Aires are a good testbed for this model, and its extension to multiple workers.

7 Related Work

As in all grid-based or metacomputing projects and research, the goal of XtremWeb is to transparently exploit networked resources on a large geographic scale through the Internet. Contrary to most projects, it does not want to exploit these resources as a giant distributed computer.

The traditional execution model of MPP is message-passing. MetaComputing or Grid-enabled infrastructures, such as Atlas [5] Globus [10] and Legion [8] extend this model to the world scale. They target tightly coupled computations, even if these cannot be as fined-grained as in a MPP context. In such computations, the remote resources invoked by the application can, and probably will have to, communicate between each other. Thus these projects have developed their own communication environments. For instance, in the canonical model of Global Computation presented in [2], communication and queuing delays are considered only between the clients and the server, and not between clients. However, in these infrastructures, clients are allowed to unlimited access to the computing resources, which is a push model.

The XtremWeb architecture differs from the various previous projects in two points. The first one is that it plans to be a multi-application environment, allowing for multiple different multi-parameter applications to run simultaneously. The second difference is that it targets high performance applications, with relatively coarse granularity.

8 Conclusion

In this paper, we have described the main design decisions about XtremWeb, a platform dedicated to study the capabilities of Global Computing.

Global computing system are much more challenging than existing cycle stealing systems which only run inside a LAN environment. We have presented the main issues. They are related to the typical number of machines involved in a Global computing system, the security and protection of the servers and the workers, the MTBF of the workers and the dynamicity of all these parameters.

Design decisions first concern the application domains considered for XtremWeb. XtremWeb is dedicated to *embarrassingly parallel* or *multi-parameter* applications. The other design decisions which are 1) Pull and Steal model, 2) One-sided communication and 3) Native code execution correspond to a) the specificities of cycle stealing on WAN environment and b) high performance requirement. The first implementation of XtremWeb relies on Java based coordination and coupling.

Early experiments have shown the necessity of a performance analysis methodology reflecting the features and interactions of this new "parallel architecture" components, servers, networks and workers.

Finally, we have described the implementation of Aires on top of XtremWeb. Aires is a large-scale end-user application used in astrophysics.

Our immediate work is to complete the first XtremWeb version, which will be available soon. The next work is to define the relevant performance parameters, which implies to separate the impact of the network, the OS, and the Java infrastructure, and to define benchmarks that can measure these parameters across various configurations. With these two tools, we plan to build a performance model and to experiment on static scheduling. Finally, we will look for other applications.

The project development can be followed from the project web site:
http://www.XtremWeb.net

References

1. Abramson, D., Buyya, R and Giddy, J. "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid, *International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia'2000)*, Beijing, China. IEEE Computer Society Press, USA, 2000.
2. K. Aida, U. Nagashima, H. Nakada, S. Matsuoka and A. Takefusa. Performance Evaluation Model for Job Scheduling in a Global Computing System. In *7th IEEE Int. Symp on High Performance Distributed Computing*, pages 352–353, 98.
3. Anderson D., Bowyer S., Cobb J., Gedye D., Sullivan W. T. and Werthimer D. A New Major SETI Project Based on Project Serendip Data and 100,000 Personal Computers. in *Astronomical and Biochemical Origins and the Search for Life in the Universe*, Proc. of the Fifth Intl. Conf. on Bioastronomy, 1997
4. T. E. Anderson R. Wahbe, S. Lucco and S. L. Graham. Efficient Software-Based Fault Isolation. In *Symp. on Operating System Principles*, 1993.
5. Baldeschwieler J. E., Blumofe R.D. and Brewer E.A.. Atlas: An Infrastructure for Global Computing. in *Proc. of HPCN'95, High Performance Computing and Networking Europe*, Lecture Notes in Computer Science 918, pp. 582-587, Milano, Italy, May 1995
6. J.Basney and M.Levy. *Deploying a High Throughput Computing Cluster*, volume 1, chapter 5. Prentice Hall, 99. R.Buyya Ed.
7. The Pierre Auger Observatory Cronin J. (University of Chicago) and Watson A. (University of Leed) http://www.auger.org.
8. A. S. Grimshaw and W. A. Wulf. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, Volume 40, Number 1, Pages 39–45, January 1997.
9. G.Fedak. Exécution délocalisée et Répartition de Charge : une Étude Expérimentale. In *RenPar'2000*, 2000.
10. I.Foster and C.Kesselman. The Globus Project: a Status Report. in *Futur Generation Computer System*, 40:35–48, 99.
11. J. Maasen, R. van Nieuwpoort, R. Veldema, H. E. Bal and A. Plaat. An Efficient Implementation of Java's Remote Method Invocation. In *Proc. ACM Symposium on Principles and Practice of Parallel Programming.* May 1999.
12. Rosenberg A.L.. Guidelines for Data-parallel Cycle-Stealing in Networks of Workstations. *Journal of Parallel and Distributed Computing*, 59:31–53, 99.