Available online at www.sciencedirect.com
SCIENCE DIRECT®
Future Generation Computer Systems 21 (2005) 417–437
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/future

Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid

Franck Cappello [a,*], Samir Djilali [a], Gilles Fedak [a], Thomas Herault [a], Frédéric Magniette [a], Vincent Néri [b], Oleg Lodygensky [c]

[a] *INRIA, LRI, Université de Paris Sud, Orsay, France*
[b] *LRI, Université de Paris Sud, Orsay, France*
[c] *LAL, Université de Paris Sud, Orsay, France*

Abstract

Global Computing systems belong to the class of large-scale distributed systems. Their properties high computational, storage and communication performance potentials, high resilience make them attractive in academia and industry as computing infrastructures in complement to more classical infrastructures such as clusters or supercomputers. However, generalizing the use of these systems in a multi-user and multi-parallel programming context involves finding solutions and providing mechanisms for many issues such as programming bag of tasks and message passing parallel applications, securing the application, the system itself and the computing nodes, deploying the systems for harnessing resources managed in different ways. In this paper, we present our research, often influenced by user demands, towards a Computational peer-to-peer system called XtremWeb. We describe (a) the architecture of the system and its motivations, (b) the parallel programming paradigms available in XtremWeb and how they are implemented, (c) the deployment issues and what mechanisms are used to harness simultaneously uncoordinated set of resources, and resources managed by batch schedulers and (d) the security issue and how we address, inside XtremWeb, the protection of the computing resources. We present two multi-parametric applications to be used in production: Aires belonging to the high energy physics (HEP) Auger project and a protein conformation predictor using a molecular dynamic simulator. To evaluate the performance and volatility tolerance, we present experiment results for bag of tasks applications and message passing applications. We show that the system can tolerate massive failure and we discuss the performance of the node protection mechanism. Based on the XtremWeb project developments and evolutions, we will discuss the convergence between Global Computing systems and Grid.

Keywords: XtremWeb; Large-scale distributed systems; Global Computing systems

1. Introduction

In September 1999, preliminary versions of Seti@home were running on some of the PhD student PCs at LRI (Computer Science Laboratory of Paris South University). For a team developing research on high-performance computing (HPC), traditionally using vector supercomputers or cluster of multi-processors, the architecture of Seti@home and its applicability to a large number of applications and users (not only contributors but also clients of

* Corresponding author.
E-mail address: franck.cappello@lni.fr (F. Cappello).

0167-739X/$ – see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.future.2004.04.011

the system) were questionable. All members of our team were more than skeptic on the usefulness of this approach, called Global Computing (GC). Moreover, the Grid was emerging as a promising idea, born as a result of profound reasons and pushed by top level research teams. However, the skepticism about GC and the alternative that it could represent to Grid were at the origin of the XtremWeb project: understanding its limitations and providing solutions to overcome some of them were really exciting research challenges. In this paper we will present the research conducted during these 4 years of work. Even though we do not explicitly mention them, user demands have significantly influenced the research in XtremWeb. Participant PC security, message passing using standard libraries and deployment techniques have all been derived partly from user suggestions. The system architecture itself has evolved from a monolithic organization to a layered one, in response to the diversity of actual and potential usage contexts.

1.1. Large-scale distributed systems

What makes a fundamental difference between pioneer GC systems such as Seti@home, Distributed.net and other early systems dedicated to RSA key cracking and former works on distributed systems is the large scale of these systems. The notion of large scale is linked to a set of behaviors that has to be taken into account if the system should scale to a high number of nodes. An example is the node volatility: a non predictable number of nodes may leave the system at any time. Some researches even consider that they may quit the system without any prior mention and re-connect the system in the same way. This behavior raises many novel issues: under such assumptions, the system could be considered as fully asynchronous (it is impossible to provide bounds on message transits, thus impossible to detect some process failures), so as it is well known [23] no consensus could be achieved on such a system. Another example of behavior is the complete lack of control of nodes and network. We cannot decide when a node contributes to the system nor how. This means that we have to deal with the in place infrastructure in terms of performance, heterogeneity and dynamicity but also that any node may intermittently inject Byzantine faults.

1.2. Global Computing systems

GC systems have emerged while the HPC community was considering clustering and hierarchical designs as good performance-cost trade-offs. They essentially extend the notion of cycle stealing beyond the frontier of administration domains. The very first paper discussing cycle stealing [50] presented the Worm programs and several key ideas that are investigated currently in autonomous computing (self replication, migration, distributed coordination, etc.). It is interesting to notice that this paper has been partially inspired by a classic science fiction film called The Blob. Due to its high attractiveness, cycle stealing has been studied in many research projects like Condor [33], Glunix [26] and Mosix [8] to cite a few. A first approach to cross administration domains was proposed by Web computing projects such as Jet [37], Charlotte [9], Javeline [15], Bayanihan [44], SuperWeb [4], ParaWeb [13] and PopCorn [35]. These projects have emerged with Java taking benefit of the virtual machine properties: high portability across heterogeneous hardware and operating systems, large diffusion of virtual machine in Web browsers and a strong security model associated with bytecode execution. Performance and functionality limitations are some of the fundamental motivations of the recent generation of GC systems like COSM [2], BOINC [6] and XtremWeb [21].

The high-performance potential of GC platforms has also raised a significant interest in the industry. Companies like Entropia, United Devices, Platform, Grid Systems and Datasynapse propose GC middleware often known as Desktop Grid or PC Grid systems. Performance demanding users are also interested by these platforms, considering their cost-performance ratio which is even lower than the one of clusters. Thus, several Desktop Grid platforms are daily used in production in large companies in the domains of pharmacology, petroleum, aerospace, etc.

GC systems share with Grid a common objective: extend the size and accessibility of a computing infrastructure beyond the limit of a single administration domain. In Ref. [24], the authors present the similarities and differences between Grid and Global Computing systems. Two important distinguishing parameters are the user community (professional or not) and the resource ownership (who own the resources and who is using them). From the system architecture per-

spective, we can consider for the rest of the paper two main differences: the system scale and the lack of control of the participating resources. These two aspects have many consequences at least on the architecture of system components, the deployment methods, programming models, security (trust) and more generally on the theoretical properties achievable by the system.

A multi-users/multi-applications GC system would be in principle close to a peer-to-peer (P2P) file-sharing system such as Napster [20], Kazaa and Gnutella [45], except that the ultimate shared resource is the CPU instead of files. The scale and lack of control are common behaviors of the two kinds of systems. Thus, it is likely that similar solutions will be adopted for their fundamental mechanisms such as lower level communication protocols, resource publishing, resource discovery and distributed coordination.

The rest of this paper is structured as follows. The next section presents XtremWeb GC system architecture and implementation issues. Section 3 discusses programming interfaces for GC systems and presents results of some experiments including execution of RPC and MPI applications using XtremWeb. Security issue and how participating nodes security is resolved for XtremWeb are discussed in Section 4. Some experimental results obtained on a testbed gathering hundreds PCs over three sites are discussed in Section 5. The last section (Section 6) discusses lessons learned from XtremWeb in the perspective of convergence between GC and Grid.

2. XtremWeb

The aim of the XtremWeb project is to investigate how a large-scale distributed system (LSDS) can be turned into a parallel computer with classical user, administration and programming interfaces possibly using fully decentralized mechanisms to implement some system functionalities. XtremWeb belongs to the more general context of Grid research and follows the standardization effort towards Grid Services [25].

We present XtremWeb-V1 in this section. This version is designed for supporting constraints imposed by LSDS like volatility, heterogeneity and security. It profits from a modular architecture based on services for more facilities in implementation and deployment.

XtremWeb, as a multi-users, multi-applications, GC project for research and production, aiming at executing external applications on participant resources, must not only provide solutions for classical LSDS issues (volatility, heterogeneity) but also complies to parallel application programming which must stay reasonable in complexity for the programmer.

Another important issue is the security. This issue is particularly difficult in the context of LSDS because it is impossible to trust hundreds of thousands resources. A first problem concerns the protection of the participating nodes. No aggressive application should be able to corrupt neither data nor system of any resource. This is particularly tight if binary applications are to be executed. A second problem linked to the lack of trust is the need for some result certifications procedure. Since there is no way to control precisely what happens on a participating resource, faulty and malicious behaviors must be detected. Section 4.1 discusses participant security.

2.1. Services

Nowadays, Grid Computing is considering the notion of services [25], a wide spread paradigm to standardize components in distributed systems. A service is an entity that must be auto-descriptive, dynamically published, creatable and destructible, remotely invoked and manageable (including life time cycle). The standardization effort also includes the use of well defined standards (WSDL, SOAP, UDDI, etc.) of Web Services [1]. A typical GC platform gathering client nodes submitting task requests to a coordinator which schedules them on a set of participating workers can be implemented in term of services: the coordinator service publishes application services and schedules their instantiations on workers; the client service requests task (association of application and parameters) executions corresponding to published application services and collects results from the coordinator service; the worker service computes tasks and sends their results back to the coordinator service. Note that the implementation of the coordinator service can rely on sub-services such as a scheduler, a data server for parameters and results, a service repository/factory which themselves may be implemented in centralized or distributed way.

2.2. XtremWeb architecture

XtremWeb follows the general vision of a LSDS turning a set of non-specific resources (possibly volatile) into a runtime environment executing services (application modules, runtime modules or infrastructure modules) and providing volatility management.

Fig. 1 presents the layers of this general architecture. This architecture considers four main layers representing a total of seven sub-layers. The role of the first layer is to aggregate non-specific resources (clusters, home PCs, PCs in LAN, etc.) for building a full, but even unstable (with a possibly volatile nodes) cluster. The second layer turns the non stable cluster into a virtual stable cluster eventually exposing to the upper layer fewer resources than actually available because of volatility management (some resources may be kept as spare ones). The third layer creates a generic GC platform. The fourth layer deploys runtime environments modules for parallel computing such as Master-Worker or MPI environments. Applications are supposed to be executed on top of the last layer.

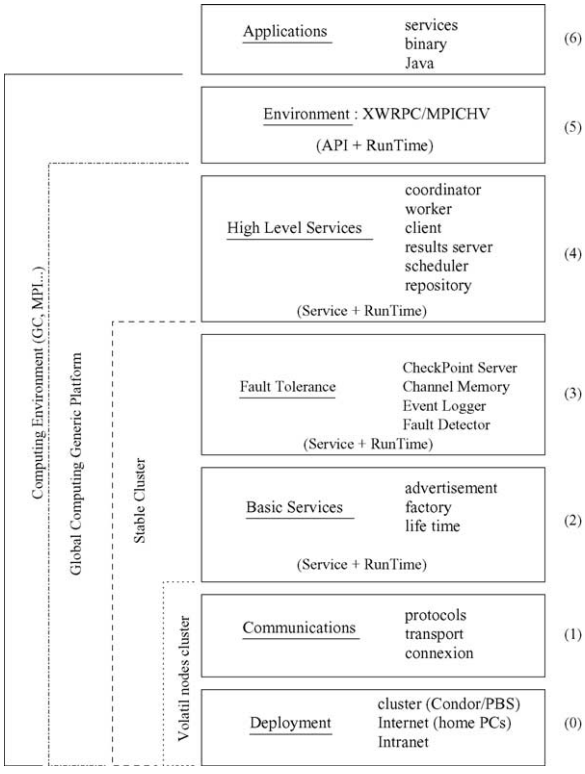

Fig. 1. A general software stack for LSDS including fault management.

The actual architecture encompasses more sub-layers. The role of the bottom most sub-layer (0) is to enable the deployment of the minimal piece of code called "Launcher" in XtremWeb or a peer in P2P environments such as Jxta in different kinds of PCs platforms (clusters, Internet, Intranet). Section 2.3.1 presents details about this layer. Sub-layer (1) encapsulates communication infrastructures allowing the communication between "launcher" or peers. Sub-layer (2) gathers infrastructure services enabling the publishing of services, service discovery, service construction, etc. Sub-layer (3) encapsulates services and specific runtimes dedicated to fault tolerance accordingly to (a) the expected final GC platform type and (b) the deployment type. Replication, message logging and fault detection services as well as coordination of fault tolerance services for implementing fault tolerance protocols are located in this layer. The next sub-layer (4) contains high-level services such as request servers, schedulers, task repositories, result servers, workers (a service working as a runtime environment for a binary or Java applications inside a PC), clients (a service enabling a user or an application to submit requests to the system). Parallel computing API and runtimes fit in the sub-layer (6). Typically the integration of programming environment with fault tolerance properties would add fault management services and runtime into sub-layer (3), API and execution runtime (launching, termination) into sub-layer (5).

The XtremWeb GC platform implements a subset of this architecture. As a GC platform, XtremWeb allows a set of clients to submit task requests to the system which will execute them on *workers*. The XtremWeb GC design follows a set of three main principles: (1) a three-tier coordination architecture connecting *client* to *workers* through a Coordination service, (2) a set of security mechanisms based on autonomic decisions, (3) a fault tolerance design allowing the mobility of *clients*, the volatility of *workers* and failure of the Coordination service.

The three-tier architecture adds a middle tier between *client* and *worker* nodes. Thus there is no direct P2P task submission/result transfer between *clients* and *workers*. The role of the third tier, called the coordinator, is (a) to de-couple *clients* from *workers* and

(b) to coordinate tasks execution on *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*, detects *worker* crash/disconnection, re-launches crashed tasks on any other available *worker*, collects and stores task results, delivers task results to *client* upon request. The *coordinator* is not necessarily implemented by a centralized (even replicated) node but can rely on distributed services deployed on participating nodes along with other services such as *clients* and *workers*.

2.3. Some key elements of XtremWeb implementation

XtremWeb is the result of several implementation decisions directly deriving from the constraints related to the user environment and the necessity to progress step by step towards the system organization described in the previous section. As a consequence, some parts of the system are currently implemented in a centralized way. We will describe several layers of the system stack (Fig. 1): deployment (sub-layer (0)), communications (sub-layer (1)), basic services (sub-layer (2)), GC services (sub-layer (4)). Some elements of sub-layer (3) will be detailed in Sections 3 and 5 related to fault-tolerant programming environments.

2.3.1. Deployment

Deployment is one of the first difficulties encountered when installing a GC system. Deployment concerns the installation but also the upgrade of the system components. Depending on the applications and users, the GC system can be installed on home PCs connected to the Internet by cables or DSL, on PCs of student classrooms in Universities and Schools, on PCs of a company connected to a private network or on clusters of PCs. Altogether, these deployment cases cover a large spectrum of installation procedures, security configurations/settings, system administrations and intrusiveness limits. In XtremWeb, deployment mainly concerns *worker* and *client* executed on participating PCs since the *coordinator* is currently executed by a single machine that could be installed specifically.

We will not detail in this section the deployment issues in the context of home PCs, classroom PCs and company private network of PCs since many techniques exist for installing and upgrading software on individual PCs or set of PCs (automatic installation/upgrade from a web server, from a distribution server, from CDROM, etc.).

A more interesting deployment case concerns PCs in clusters. While originally GC systems were developed for taking advantage of unused CPU cycles, through cycle stealing, they appear to provide a light weight alternative to classical Grid infrastructures for harnessing the capacities of clusters belonging to different administration domains. Using PCs of clusters should follow the security and resource management policy of the cluster administrator. In the general case, cluster accesses are granted from user account basis and task submissions go through a batch scheduler. Instead of accessing the clusters, the user logs onto the *coordinator* which manages a community of users. The cluster administrators open a generic Grid account only known by the *coordinator* administrator. User can only submit tasks through a batch scheduler interface provided by the *coordinator* (they have no direct access to the clusters). All user tasks are submitted by the *coordinator* to the cluster batch scheduler, using its proper account, as a representative of the user. The coordinator keeps track and logs the (task, user) doublet so that if a user generates a security issue on one of the clusters, appropriate actions (revocation, etc.) can be taken. There are two ways for interfacing the coordinator and the clusters for task submission. The first one consists in running a daemon on the cluster front-end which will be responsible for (1) downloading applications code and task parameters, (2) submitting the corresponding tasks, using the batch scheduler commands of the cluster, (3) forwarding/translating the control commands (kill, status, etc.) issued by the coordinator to the batch scheduler, (4) forwarding back the batch scheduler responses to the commands, (5) notifying the task terminations to the coordinator, (6) transmit the results from the cluster to the coordinator and (7) cleaning the local directory of the files related to the user tasks. The translation interface between the GC system and batch scheduler encounters the same heterogeneity problem than in computational Grids: many different kinds of batch schedulers are used on clusters and there is no standard interface. A second way of submitting user tasks is by launching a set of GC *workers* as tasks of the cluster batch scheduler. Once launched, the workers connect the GC *coordi-*

nator for requesting tasks. This approach allows installing a simple daemon on the cluster front-end simply launching the number of workers required by the coordinator. All the complexity of application codes and parameter transfers, status and termination notification, result transfer and local directory cleaning is the responsibility of the GC workers. This reduces the adaptation complexity of the interface between the GC system and a specific batch scheduler to the development of a set of few commands.

2.3.2. Communications

Communications between the different parts of XtremWeb include remote procedure call (RPC) messages and data transfers. XtremWeb communication architecture relies on three protocol layers. The first level "connection" is dedicated to enable connection between the entities possibly protected by firewall or behind a NAT or a proxy. The second level "transport" is responsible for reliable and secure message transport. The third level "protocol" gathers several flavors of RPC API.

Firewall bypassing is a main role of the connection layer. A standard firewall configuration stops inbound communications, except on some well defined host/port, and generally allows outbound ones. As XtremWeb is currently centralized, firewall bypassing can be done if the *coordinator* is reachable by other parties (i.e. if coordinator inbound communications are allowed). Communication channels are then never initiated by the *coordinator*, but by *clients*, *workers* or any other party; channels are used to transmit messages to and from parties of a communication. This implies that no party except the *coordinator* needs to be behind an opened firewall for inbound communications. Communications are possible if the *coordinator* opens dedicated ports and implements specific protocols. For instance, JavaRMI needs the rmiregistry port and XML-RPC the HTTP one; any other binding needs its own specific port and protocol implementation.

Any other connection-oriented middleware can be used for a more distributed implementation. Jxta would typically provide the required features for enabling communications between entities.

The transport layer relies on TCP/IP. Security can then be achieved with standard SSL transport by encrypting the communications and forcing authentication so that communications occur between trusted parties only.

XtremWeb has bindings for several RPC flavors (*JavaRMI* and *XML-RPC*), based on TCP-IP, and provides necessary tool, *XWIDL*, to bind others such as SOAP. It is out of the scope of this paper to discuss technology details, but we can note some clues to choose one or another. JavaRMI is the standard RPC mechanism for Java language which is platform independent, but can hardly inter-operate with other programming languages. XML-RPC (and hence SOAP), which describes remote procedure call messages in XML, transported on HTTP, is programming language independent but quite heavy and verbose. These technologies have a message size limit (i.e. few hundreds kilobytes) and cannot manage huge messages for data transfers, such as task data (parameters or results) which may be several megabytes large, and are then transferred as TCP packets.

2.3.3. Basic services

Because of the scale and dynamicity of LSDS, a client cannot identify the *worker* (or the set of *workers*) that can execute its task requests. Like for other dynamic distributed systems, several basic Services should be provided to enable the execution of client requests: a resource discovery allowing a client to discover available services, a factory allowing dynamic instantiation of services on workers, advertisement facilities giving information about available services, and the life time management to control service termination.

A typical service call follows several steps: the resource discovery engine is invoked to return a factory address being able to realize a service instantiation. When the service is instantiated, the factory returns the hosting machine address where the service can be called. In a GC platform, this last step is slightly modified since service is subject to be transparently hosted in several places during its life time (see Section 2.3.4). Finally, the service stays alive until it detects a termination condition (for example, it is not used any more or it receives a termination signal).

Several distributed system architectures and implementations are providing a subset of or all these services. Popular examples are OGSA [25] for Grid environments and Corba [36] for object-oriented distributed environments.

These basic services do not provide all mechanisms required for running a GC application. Other higher level services are necessary such as the ones described in the next section.

2.3.4. XtremWeb services

The first implementation of XtremWeb considers three main services: *client* which submits requests, *worker* which executes them and *coordinator* which plays the role of intermediary between *clients* and *workers*. In this version of XtremWeb, *coordinator* encapsulates different services (scheduler, results server, applications repository).

2.3.4.1. Coordinator architecture. The middle tier implements a set of coordination services which in principle could rely on distributed architecture. For example, the resource discovery service could rely on distributed hash tables like CAN [39], CHORD [52], PASTRY [40] and TAPESTRY [53]. However, we choose to implement all these services in a centralized way for three reasons: (1) to ease system development and debug, (2) there are few results concerning the performance comparison between centralized, hierarchical and fully distributed implementation of key services such as resource discovery for example and (3) because of a theoretical uncertainty. Currently there is no theoretical result about the fundamental classification of P2P systems as distributed systems. It is still uncertain that their two main behaviors: (a) Internet as the communication network and (b) volatility of participating nodes potentially without disconnection notification; make them fall into the category of asynchronous distributed systems. Without this result we cannot decide if we can rely on consensus or not in developing distributed services.

The *coordinator* in XtremWeb is composed of three services: the applications/services repository, the scheduler, and the result server (sub-layer (4)). These services work altogether around a tasks pool maintained consistent by a task state graph.

The applications/services repository and the scheduler implement the three minimal basic services. The repository provides an *advertisement service* by publishing services and applications and making them available to clients through standard communication ports (i.e. Java RMI, XML-RPC). The scheduler manages the XtremWeb *service factory* by instantiating services and applications on workers, and manages their *life cycle*. It starts their execution on workers when clients submit jobs, stops them as expected (on client demand or accordingly to their life cycle), and corrects GC faults, if any, by finding available workers to re-launch them. Currently, the scheduler implements the pull model of tasks allocation (i.e. tasks are allocated on workers, on demand, following their initiative). Finally, workers deposit results on the result server.

Services and applications are removed on demand (the platform makes no assumption and continues to publish and distribute them until they are discarded by client intervention) or by an automatic LRU policy.

2.3.4.2. Worker architecture. 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 may be in one of the three different states: *ready* to be computed, *running* and *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 results file to the result server; all tasks are in this last state at the end of execution and are removed only upon coordinator notification to ensure results are safely saved to the result server. The communication manager ensures communications with other entities; it downloads task files (binaries and parameters) and uploads results files, 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 (i.e. standard input and output, 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 new or modified files in the directories structure and updates task status on disk. If the computation is interrupted for any reason (i.e. if the worker is stopped by mouse/keyboard activity),

the interrupted task will be recreated from the information store on the disk and restarted on next worker execution. 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 (1) send the results again in case the upload went wrong or (2) definitively remove the task. On worker starts, any interrupted task is re-queued for execution, and any pending results are retrieved to be either re-sent to result server or removed from disk.

These two mechanisms ensure fault tolerance by (1) restarting computation of any interrupted tasks and (2) ensuring results uploads even on case of result server failure.

2.3.4.3. Client architecture. The client is an intermediary between user's application and the GC system. It is implemented as a library plus a daemon process. The library provides an interface allowing a dialog between the application and the *coordinator*. The basic actions ensured by the *client* are identification, tasks submission and results retrieval. To enhance fault tolerance, the client daemon implements two mechanisms: (1) local logging of exchanged messages by saving on disk all requests descriptions and exchanged files, and (2) synchronization with coordinator at connection/re-connection time by getting its previous requests and results from the *coordinator* which saves this information on its local disk. The first mechanism insures recovery on the same machine in case of fault and the second one makes possible to stop and re-launch the client (user's application) on any machine. These two features allow a high mobility of the client.

3. Parallel programming API

Ideally, GC systems architects should provide programming environments (interfaces and runtimes) close to the ones used on parallel computers with an automatic management of the specific behavior of GC platforms: (a) implementing classical and well known programming paradigms, (b) exposing standard programming interface, (c) removing the burden of heterogeneity management, (d) providing efficient execution and high performance, (e) hiding faults detection and management.

Pragmatic implementations would certainly relax some of these highly desirable properties but at the cost of a potentially high deviation from the standard interface and a higher complexity for the programmer. For example, it is attractive for the system developers to transfer the dynamicity and fault management to the programmer by augmenting a standard programming interface with function return codes exposing the runtime states and specific functions for managing them. FT-MPI is an example of this approach [18].

In XtremWeb, we decided to provide standard programming interfaces, placing all the management of GC platform specificities in the runtime. We follow the layered architecture presented in Fig. 1: we built on top of an instable and dynamic distributed system a virtualization layer providing to higher layers the illusion of a stable and static execution platform.

3.1. Concurrent RPCs

The concept of remote procedure call [30] has been used for a long time in distributed computing as it provides a simple way to allow communication between distributed components. A significant portion of scientific applications can be programmed using the concurrent RPC programming style to implement Master–Worker or work-flow-based applications. In these applications, a *client* may launch a set of non-blocking RPCs to different servers, leading to a concurrent execution. The client controls the progress of the execution, detects task completion, manages parameters/results dependencies, launches eventual subsequent concurrent executions. CondorMW [27] was one of the first programming environments for programming Master-Worker applications. The RPC programming style encounters a large popularity for the Grid and several programming environments have been proposed. The most known are GridRPC [49] and OmniRPC [46].

GridRPC is a proposal to standardize a remote procedure call mechanism for Grid computing. Two different Grid computing systems, NetSolve [14] and Ninf [47], propose implementations of this standard. Netsolve and Ninf have not been designed to handle the volatility of nodes in LSDS systems, even though basic fault tolerance has been investigated for Netsolve [38]. OmniRPC is another proposal of RPC-based programming environment for the Grid.

In the RPC implementation for XtremWeb, called XWRPC [17], the client automatically translates a RPC call into a task manageable by the coordinator. When it gets the result file back, the client extracts from it the output parameters. XWRPC provides blocking and non blocking RPC calls with some associated Wait functions.

Most of the previous works on RPC have focused on the development of high-performance RPC mechanisms and RPC for the Grid. Little attention has been paid concerning fault tolerance in cluster, Grid and GC contexts. However, fault tolerance has been deeply studied for object-oriented distributed systems and especially in the context of the Corba middleware. One elegant proposal for FT-Corba implementation relies on a three-tier architecture, close to the one of XtremWeb, making the middle tier the corner stone of the fault tolerance protocol [7]. This implementation of FT-Corba assumes that its middle tier is deployed on a pseudosynchronous system. This ensures the coherence of the middle tier replication.

Our implementation of fault tolerant concurrent RPC follows some severe constraints: automatic and complete transparency to the client, on a fully asynchronous system. Thus we restrict our studies to stateless executions on the worker side. RPC fault tolerance aims to certify that all RPC calls succeed. Following the architecture presented in Fig. 1, sub-layers (4) and (5) use an imperfect failure detector of sub-layer (3) to ensure each RPC call success. In such a system, it is impossible to design a perfect failure detector (which suspects every and only dead processes); thus, our failure detector may wrongly suspect some processes, resulting in an over submissions of RPC calls, which are harmless for the system (but degrade performances).

For performance reasons, replication is implemented for all parts, but these replicas are not necessarily coherent to each others. Using this fault tolerance scheme, any entity (client, coordinator or worker) of the system can disappear without affecting the integrity (but the performance) of a RPC. The coordinator detects workers crash/disconnection by time-out mechanism and re-schedules their allocated tasks on other available workers using logged messages.

To highlight the fault tolerance properties of XWRPC, we present the result of running NAS NPB 2.3 EP Benchmark (decomposed in 100 tasks, spending 15 s each one)—class C on the XtremWeb platform using AMD 1.5 GHz (500 MB RAM) PCs connected by a switched Ethernet 100 Mb s^{-1} network. Fig. 2 shows the execution times of EP benchmark—class C on 16 processors (workers) when transient or definitive faults are artificially generated.

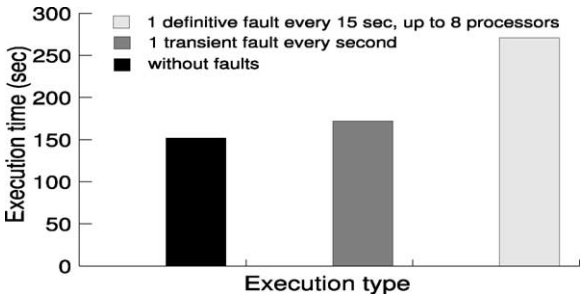

Fig. 2. Execution of EP benchmark in faulty environment.

For transient faults, a worker disappears every second, for few seconds (2–3 s) and then re-connects the system. For definitive faults, when a worker crashes, it never re-contacts again the coordinator during the experience (this implies that after detecting this fault, the coordinator has to re-schedule the task to another available worker). A crash happens every 15 s up to eight workers (loss of 50% of workers). Fig. 2 demonstrates that the application can survive frequent transient and definitive faults. These faults are responsible for respectively 12 and 78% of execution time increases. The low degradation of performance for transient faults is mainly due to the task logging mechanism on the worker which saves its last running task (on disk), and re-executes it when it re-connects the system. If we take in account that these faults are handled automatically by the system and no programming efforts are needed from the programmer, then this overhead stays reasonable. We are currently testing the system tolerance to client and coordinator faults. Results of this experiment will be published in another paper.

3.2. SMPD through fault-tolerant MPI

Users of high-performance computing platforms are familiar with symmetric message passing and their applications often use MPI [51] as the message passing

library. The high volatility of nodes in LSDS implies the use of a fault-tolerant MPI implementation. The way how faults should be managed in the context of MPI is still an open issue [28]: (a) the programmer of the application may save periodically intermediate results on reliable media during the execution in case of an entire restart, (b) the functions of the MPI implementation may be augmented to return information about faults and accept communicator reconfiguration [19] or (c) the MPI implementation hides the faults to the programmer and the user by providing a fully automatic fault detection and recovery. The latter approach, while interesting for the end user, suffers either from limited fault-tolerant capabilities or high resource cost. Examples of such automatic fault-tolerant MPI implementations are based on the optimistic or causal message logging approach. While in theory these protocols may tolerate any number of faults if augmented by appropriate mechanisms, none of their existing implementation tolerates more than one fault, involving the restart of the full system in case of multiple faults. Examples of automatic faults-tolerant MPI protocols that tolerate N concurrent faults of MPI processes, N being the total number of MPI processes, follow the pessimistic message logging principle (storing all in transit messages on reliable media) and thus require a large number of non-computational reliable resources. Refs. [11,12] present extensively the related work of this domain.

To study several fault tolerance protocols for MPI, we have launched the MPICH-V project which is a research effort with theoretical studies, experimental evaluations and pragmatic implementations of a fault-tolerant MPI. A MPICH-V environment encompasses a communication library based on MPICH [29] and a runtime environment. The MPICH-V library can be linked with any existing MPI program as usual MPI libraries. In case of LSDS, the runtime (responsible for process distribution, fault detection, process restart, checkpoint scheduling, etc.) occupies several layers (at least 3 and 5) of the software stack, as seen in Fig. 1.

The library implements all communication subroutines provided by MPICH. Its design is a layered architecture: the peculiarities of the underlying communication facilities are encapsulated in a software layer called a *device*, from which all the MPI functions are automatically built by the MPICH compilation system.

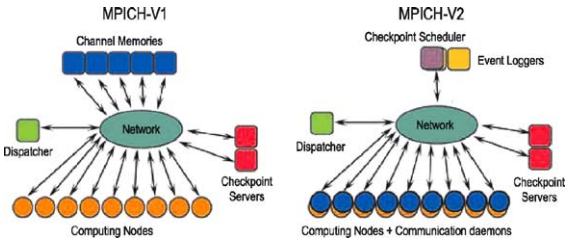

Fig. 3. General organization and components of MPICH-V1 and MPICH-V2.

The MPICH-V library is build on top of a dedicated device ensuring a full-fledged MPICH v. 1.2.5, implementing the Chameleon-level communication functions. The underlying communication layer relies on TCP for ensuring message integrity.

Two protocols, based on uncoordinated checkpointing associated with pessimistic message logging, have been implemented and compared for LSDS: MPICH-V1 and MPICH-V2. The two protocols use a reliable coordinator and checkpoint servers. Fig. 3 presents the general organization and the components used for the two protocols.

MPICH-V1 relies on the concept of Channel Memory (CM) to ensure fault tolerance. CMs are dedicated nodes providing a service of message tunneling and repository. Fault tolerance is implemented in a highly decentralized way, saving the computation and communication contexts independently. For each node, the execution context is saved (periodically or upon a signal reception) on remote checkpoint servers. A communication context is stored during execution by saving all in-transit messages in CMs. Thus, the whole context of a parallel execution is saved and stored in a distributed way. The runtime assigns CM to nodes by associating each CM to different sets of receivers. Following this rule, a given receiver always receives its messages from the same CM, called its *home CM*. When a node sends a message, it actually puts the message in the receiver *home CM*. During a restart, the process is re-executed from its last valid checkpoint image and communications are replayed until the process reaches the crash point. Emissions are simply filtered since they are already saved on the destination node CM. Receptions are re-executed contacting only the process *home CMs*. MPICH-V1 is described extensively in Ref. [11].

MPICH-V1 imposes that each message crosses one CM. This has a direct impact on performance, reducing the bandwidth and increasing the latency by a factor of 2. The number of CMs should also be significant since all receivers associated with the same CM will share its communication bandwidth. To overcome these limitations, we have designed MPICH-V2 which associates the low additional resource cost of sender-based message logging and the capacity to tolerate N concurrent faults of the pessimistic message logging strategy. The sender-based pessimistic message logging protocol assumes that the logging of messages is split in two parts. One part uses a sender-based logging method storing the messages payload within the sender on a non-reliable media. The other part (the event logger) is used to store dependency information associated to these messages and must be run on a reliable system. It stores and delivers dependency information about messages exchanged by the computing nodes. On restart, a process is re-executed from its last valid checkpoint image and the event logger sends it the identity of the senders that have sent messages between the last process checkpoint and the crash. These sender nodes are then notified to re-send the lost messages using their local message logs. MPICH-V2 is described extensively in Ref. [12].

At least three parameters are significant for comparing the respective merits of fault tolerance protocols: performance overhead, number of stable nodes required for high performance and tolerance to frequent faults.

We evaluate the performance using a cluster of PCs, in dedicated mode, under Linux 2.4.18. The cluster consists in two parts: 32 computing nodes (Athlon XP 1800+, running at 1.5 GHz and 1 GB of main memory), and 12 auxiliary machines (dual-Pentium III machines running at 500 MHz with 512 MB of memory) connected to a single 48-port Ethernet 100 Mb s^{-1} switch. Test programs are compiled using the PGI PGF77 compiler.

Fig. 4 presents execution time breakdown of the NAS Benchmarcks NPB2.3 CG Class A and BT Class B (two extremes of the computation to communication ratio) for three MPI implementations: MPICH-P4 (the reference implementation for TCP), MPICH-V1 and MPICH-V2.

The system setup for MPICH-V1 uses $N/4$ memory channels, N being the number of computing nodes. A single checkpoint server is used. For MPICH-V2, it uses a single reliable node, holding a single checkpoint server and a single event logger.

Fig. 4 shows that the computation times are the same for all the implementations for the two benchmarks. The poor performance of MPICH-V1 and MPICH-V2 for CG-A is explained by the communication time, which increases dramatically, due

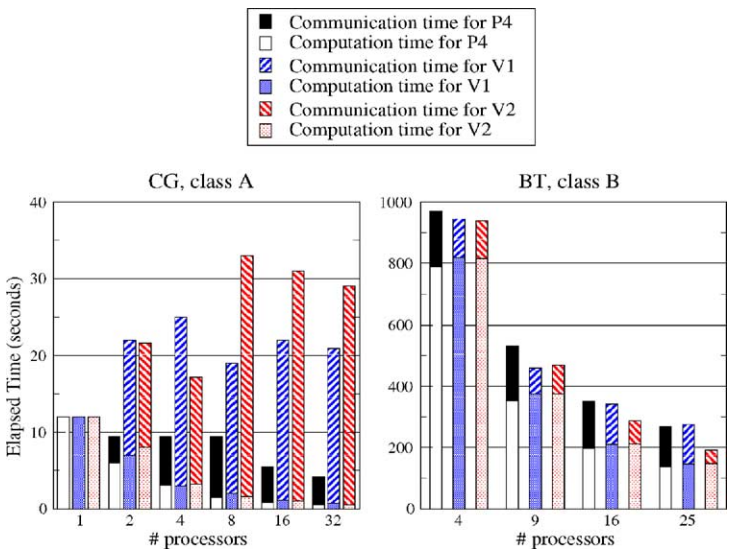

Fig. 4. Execution time breakdown of the three MPI implementations for CG-A and BT-B.

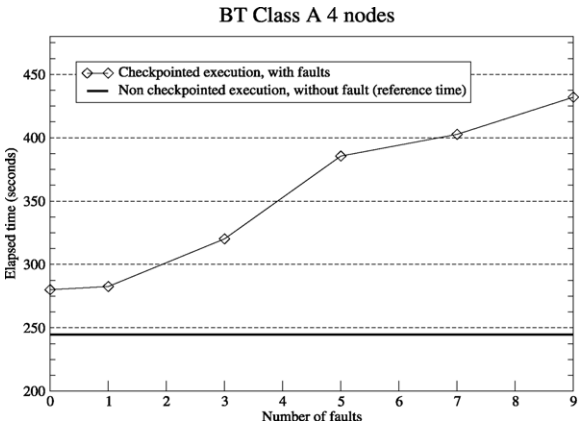

Fig. 5. MPICH-V2 performance for BT-A on four nodes when the number of faults increases during the execution.

to the overhead of the message logging protocols. MPICH-V1 outperforms MPICH-V2 on this test because the event logging of MPICH-V2 requires acknowledgment messages that significantly increase the latency for small messages. For BT-B, the communication performance of MPICH-V2 is better than both MPICH-P4 and MPICH-V1. This is due to lower bandwidth of MPICH-V1 for large messages (each message should cross a CM) and the different ways how asynchronous communications are handled in MPICH-P4 and MPICH-V2.

These results show that message logging has a non-negligible impact on performance for applications featuring short messages and low computation to communication ratio. In the other cases that are the actual targets of MPICH-V, the sender-based protocol provides better performance than the protocol using remote logging and requires much less reliable nodes (1 versus 9 for the 32 computing nodes of our experiments).

Fig. 5 presents the MPICH-V2 execution time for BT Class A using four computing nodes and a single reliable node for executing the checkpoint server and the event logger, when faults occur during the execution. For this test, (a) the checkpoint of a node immediately follows the one of another node, (b) we simulate faults by sending a termination signal to a randomly selected MPI process, (c) the execution is restarted immediately from the checkpoint image provided by the checkpoint server.

Fig. 5 demonstrates (1) the low overhead of the checkpoint system when no fault occurs (less than 20% of the execution time), (2) the smooth degradation of the execution time according to the number of consecutive faults, and (3) an execution time lower than twice the reference execution time (without fault) when nine faults occur during the execution. This last test clearly highlights the fault tolerance properties of MPICH-V2 which tolerates up to 1 fault every ∼50 s. Similar results have been obtained for MPICH-V1 [11]. Thus the difference between the two protocols comes from their performance overhead and the number of required stable nodes. MPICH-V2 outperforms MPICH-V1 on these two parameters.

4. Security

The security issue for a multi-participant GC system can be divided into four parts: the data integrity/privacy, the application result correctness, the infrastructure integrity and the integrity/privacy of the participating computers resources. Several works have been presented for the first three parts [16,42]. In this part we focus on the later part.

The security system must protect the participant computers from virus-like attack, including hardware alteration, configuration modification, personal files spying and worms introduction. A general approach to protect a computer running a program is to confine the code execution inside an unbreakable envelope. Sandboxing is a well known technique implementing this principle by filtering the system calls which appear to be the main security holes of recent operating systems.

According to a security policy, sandboxing neutralizes hostile behaviors, limits resource usage and prevents any attempt to exploit security holes on the host. Because it offers a complementary security mechanism providing a runtime control of the execution, sandboxing may appear as the cornerstone technology to allow a wide and safe use of LSDS systems. The most known sandbox mechanism is the Java Virtual Machine which interprets a byte-code. All interactions between the program and the system pass through the JVM which filters them. Moreover, the protection domain mechanism allows the JVM to filter the system calls in a different way in the successive phases of the program. Unfortunately, the Java machine cannot

handle easily native codes and thus does not provide a generic sandboxing solution.

4.1. Sandboxes

Three kinds of native sandboxes can be used to protect a participant machine. Several of them are based on Ptrace which is a mechanism implemented on all Posix compliant systems. The task to spy is spawned by a monitor, referenced in the kernel. Every time the task issues a system call, it is frozen and the monitor is waked-up by the system in order to check the task memory for the argument validity. If arguments are compliant with the security policy, the system call is executed by the kernel and the computation continues. In the opposite case, the task is destroyed. The main sandbox based on Ptrace is Subterfugue [34], a monitor which uses python policies. Ptrace suffers from a security hole exposed to race-condition attacks: between the monitor granting time and the system call execution time, the arguments can be changed by another thread of the task.

A more secure mechanism follows the principle of system call emulation. A virtual native machine intercepts the system calls of the programs, checks their validity and issues them on the actual system, forwarding the result to the application. This technique offers an easier control of the runtime environment, or the capacity to emulate an operating system on one another. Two projects implement this principle: User Mode Linux (UML) is a user space program acting as a Linux kernel by emulating all the system calls. VMware provides virtual machines on top of a host operating system, giving them the illusion of a direct access to the host hardware. By this way it allows the execution of a full and standard operating system inside a virtual machine running on an operating system of another kind. Virtual machines are secure since the accesses of the actual host resources are fully controlled by the virtual machine. However, this approach is slow: the time overhead for some system calls can be as large as a factor of 1000. Such overhead may be unacceptable in a high-performance GC system.

A more recent sandboxing method based on Linux Security Module (LSM) has been proposed providing an interposition mechanism directly inside the kernel. LSM is a framework in the Linux kernel which allows inserting security modules directly in codes of system calls. This solution is secure because syscall checking is done inside the kernel. It is also efficient since the execution of the verification code does not imply any context switch. As far as we know, there is no existing module specifically designed for the context of GC and P2P systems. This is the reason why we have developed SBLSM described in the next section.

4.2. SBLSM

SBLSM is a module for LSM dedicated to GC and P2P systems. The principle is to apply a security policy to a set of binaries processes. Every time a sandboxed process issues a system call, the module checks a dedicated variable which can take three different states: GRANT, the specific controls are called, DENY, the call is denied and return with an error number, and ASK, the module asks an authority (i.e. an administrator) what to do via the security device. The goal of this preliminary check is to execute the specific controls (which can be slow) only when this is necessary. If the call is granted, the specific permission verifications are called.

Currently, SBLSM provides three controls—(1) File access control: if the system call manipulates files, the module checks if these files are in its permissions list. This list includes files or directories, and can be used in a positive way (the authorized files must be in the list) or negative (must not). (2) Network access control: in the same way, the module filters the network connections with a black or a white list. This kind of limitations can be useful for communicating sandboxed applications (MPI, OpenMP). (3) Process Signal Control: the sandboxed processes cannot send signals nor Ptrace non-sandboxed processes. In order to communicate with user space, the module implements a special device which allows the modification of the filtering parameters using a simple and dedicated format. It allows the kernel to ask the user (ASK mode) for a decision. In that case, the format can be human readable (ASCII) for direct treatment or XML for automatic treatment. A special program allows the system administrator to load ASCII policies in the module via the special device. The module can be configured to work in

a strict mode where the spaces of sandboxed and normal processes are strictly disjoint (real confinement): the resources of one space cannot be accessed by the other. This mode allows protecting the data of an application from the other users of the machine.

4.3. SBLSM performance experiment

In this paragraph, we present an abstract of a performance evaluation of the SBLSM sandbox for common operations of GC and P2P systems that will be published separately.

The performance experiment uses a dedicated machine with minimal Linux system: PPro 200MHz with 2.4.18 kernel and 2.4.18-lsm1 patch.

For this experiment, we use synthetic benchmarks with a huge number of system calls. Indeed, there is no slowdown on calculus because the interposition cost is only on the system calls. We present two typical tasks of GC and P2P systems: (a) downloading a big file and (b) expanding a big archive. For the first benchmark, we use the wget program to download a 100 MB file over a 100 Mb s^{-1} LAN. For the second benchmark, we untar the Linux kernel archive (40 MB), which creates 14 313 files. We measure execution time for three configurations: (1) without security module, (2) security module is loaded but the program is not registered as a sandboxed one, and (3) security module is loaded with a basic security policy using white listing of authorized system operations, and the program is sandboxed.

The Fig. 6 shows the result of this experiment: the overhead of the sandbox is low, for non-sandboxed as well as for sandboxed processes, even between 1 and 6% when a massive number of system calls are issued. We believe that in the absence of standard and high-performance virtual machines, an interposition technique located inside the kernel, such as the one based on LSM, provides an attractive performance/security trade-off.

	wget	tar
without SBLSM	~ 26.51s	~ 95.95s
Outside the sandbox	~ 27.05s	~ 96.58s
Inside the sandbox	~ 28.64s	~ 97.70s

Fig. 6. SBLSM performances on synthetic benchmarks.

5. Large-scale experiments using XtremWeb

The purpose of this section is to present some experiments about deploying scientific application over three clusters managed locally with different systems. XtremWeb is used to aggregate resources of these different sites with minimal effort. The applications discussed in the following paragraphs are programmed using the XWRPC interface.

5.1. Understanding protein folding by mutations

To demonstrate the full system, we ran a biomolecular application for the IBBMC (Molecular and Cellular Biochemistry and Biophysics Institute) Laboratory at Paris South University (France) which research interests include understanding protein dynamic structure parameters that affect stability and activity of proteins.

The general context is the understanding of stability and expression parameters of proteins activity from the analysis of their dynamic properties and their structure. The main goal is to evaluate the stability and perturbing factors of the protein folding by mutations. Molecular modeling is used to explore the conformational possibilities of macromolecules at a time-scale lower than 1 ns. The application consists in a multi-parameters computation requiring a large set of independent tasks. This is a four-step process. The first step generates n starting conformations along coordinate of interest. The second step performs m constrained molecular dynamics simulations for each starting conformation ($n \times m$ workers). The third step gathers statistics and the final step computes free energy profile.

5.2. Understanding high-energy cosmic rays

The aim of the Pierre Auger Observatory is to detect showers produced by the interaction of cosmic rays of energy greater than 10^{19} eV with the atmosphere. In order to determine the origin of these cosmic rays, their direction and energy must be measured with accuracy. Their nature (photons, protons or nuclei) must be known too.

These measurements are based on the properties of the secondary particles of the shower reaching the ground (number, position, energy, nature and mean arrival time) and on the study of the nitrogen fluo-

rescence generated by these particles through the atmosphere. However, statistical fluctuations in the development of an air-shower exist. For that reason, the data analysis needs a large number of simulated air-showers, with a good accuracy, to study these fluctuations. As duration of one simulation is about 10 h, the needed simulation computing time is estimated at 10^6 h for all the experiment.

The Aires [48] (Air-shower Extended Simulations) is one of the main program of simulation used by the Auger collaboration; it is already used in Computing Center of the IN2P3 at Lyon. As computer technologies evolve, it appeared that Aires simulations can now fit with the capabilities of GC platforms.

5.3. Testbed

Fig. 7 presents the testbed used for experiencing XtremWeb at a significant scale.

Simulation computations are distributed over three different sites: two sites in France, one at the LRI and the other at Grenoble, and one site in Wisconsin, USA. The coordinator runs on a dedicated machine at LRI. Workers run on different sites, managed by batch schedulers. For these experiments, XtremWeb workers are deployed as tasks submitted to the batch schedulers. Once launched, they pull tasks to XtremWeb coordinator and work as if they were running on individual PCs. Grenoble site uses PBS [22] and Wisconsin and LRI sites use Condor [32]. As these two batch systems use different resource allocation policies, no prediction can be made about how and when our workers are scheduled. Because these sites cannot be used in dedicated mode, experiments cannot be made in the same experimental conditions. This is the reason why there is a specific configuration for each experiment.

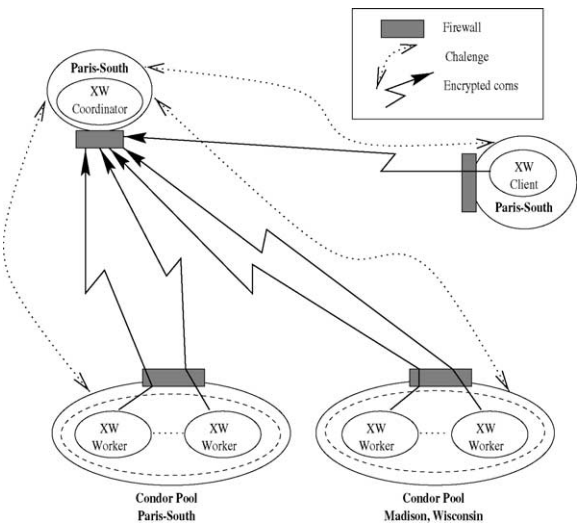

Fig. 7. An XtremWeb testbed used for testing XtremWeb using the AIRES application.

Table 1
CPU provided by different domains

Exp.	Wisc		LRI (1800 MHz)	Total
	600 MHz	900 MHz		
LRI			30	30
Wisc	61	104		165
W + LRI	50	73	9	132

Table 1 summarizes the host types used for each experiment of the biochemistry application.

For the HEP experiment we have made five experiments: WISC-97 (97 processors at Wisconsin), WL-113 (113 processors at Wisconsin and LRI), G-146 (146 processors at Grenoble), WLG-270 (270 processors at Wisconsin, LRI and Grenoble) and WLG-451 (451 processors at Wisconsin, LRI and Grenoble).

5.4. Evaluation

In this section we present selected results from the Auger experiment. Fig. 8 shows resource utilization for each experiment.

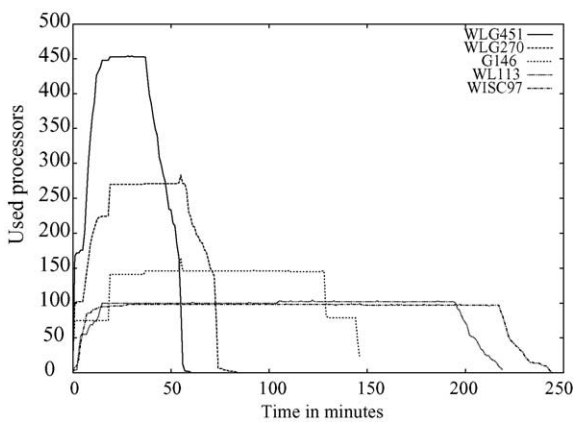

Fig. 8. Processors utilization.

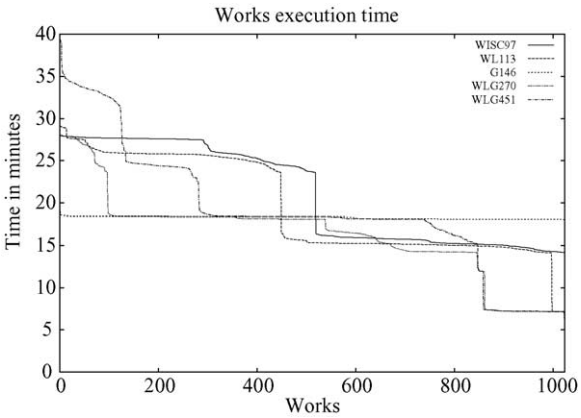

Fig. 9. Task execution times sorted left to right in a decreasing order.

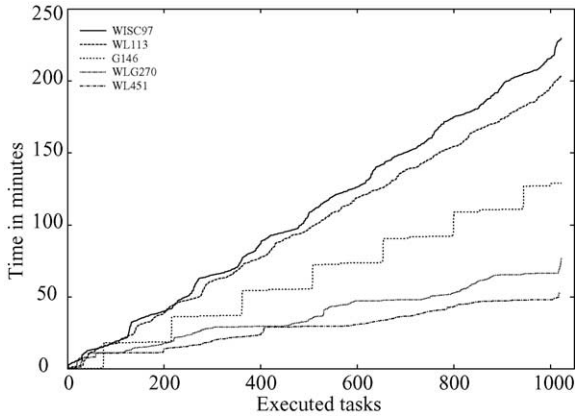

Fig. 10. Tasks results arrival time.

All curves show three different phases. The first is the warm-up phase where a growing number of PCs are allocated; then the curves reach a plateau where all the requested CPUs are contributing to the computation. The third is a cool down phase where a rapidly diminishing number of PCs are finishing their tasks. This figure demonstrates that when an increasing number of PCs is used, the execution time decreases. Fig. 9 shows the execution time for each task, decreasingly sorted. We note the remarkable stability of the system at Grenoble (G-146 curve) where all hosts are identical (CPU, memory, etc), whereas WISC-97 clearly shows two levels corresponding to the two available host types (PIII 533 MHz and 900 MHz). The stability for this experiment is bad compared to the one found in G-146, as reflected by the curve irregularity between the two plateaus. This is due to two main factors: (1) Condor may allocate other tasks to the same resource (i.e. the same CPU), depending on the local resource management policy and (2) the network conditions (performance and congestion) are not the same between Wisconsin and LRI, and between Grenoble and LRI.

Fig. 10 presents the arrival time of the task results from the first one at the left to the last one at the right. The plateau of G-146 are explained by the identical performance of the PCs and the identical task complexity. All the PCs begin and end their task at the same time. When heterogeneity and network dynamicity increase the curves become more flat. We can notice some irregularity at the end of all curves. This is due to an unexpected phenomenon: few low performance PCs get some tasks at the end of the experiment which terminates only when these PCs return their result. A better resource management, giving the last tasks preferably to the fastest PCs, would avoid this phenomenon.

The last figure (Fig. 11) presents a fault tolerance experiment during which the network connection between the coordinator and Grenoble was stopped and restored without notice.

The figure demonstrates that the system tolerates a massive fault loosing half of the participating nodes. After the fault, the PCs at Grenoble reconnect the coordinator and obtain tasks but the task allocation

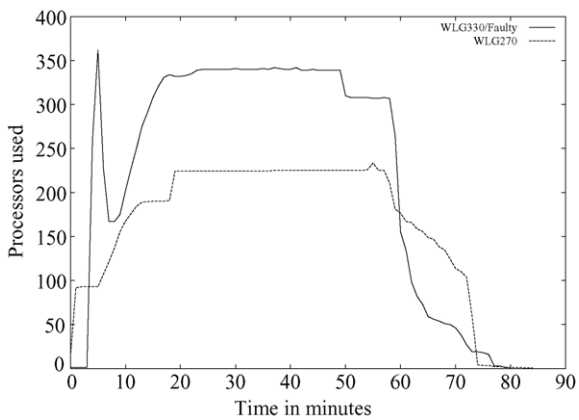

Fig. 11. Number of PCs used for the duration of the experiment. A massive fault occurs at $t = 5$ min (network disconnection with Grenoble). The network connection is restored at $t = 8$ min.

takes more time than at the beginning of the execution. This is due to the connection protocol of the worker. When the coordinator does not respond immediately, a worker increases exponentially the delay before the next connection tentative. An execution with 270 PCs is given as reference.

6. Lessons learned from the perspective of interaction between GC and grid

During the past 4 years we have investigated the issue of computing in large-scale distributed systems. All the addressed topics (deployment, programming and security) have been studied, tested and evaluated using a real platform: XtremWeb. Here are the three main learned lessons:

(I) A first intuitive idea is that harnessing the resources in clusters would lead to mechanisms closed to the ones used in Grid systems in terms of application codes, parameters and results transmission, interaction with batch schedulers and load balancing between clusters. By simply launching multiple instances of the XtremWeb worker as applications on the clusters, through their possibly specific batch schedulers, we demonstrate that (a) the transfer of codes, parameters and results can be completely handled by the GC system, removing the need to implement specific mechanisms for these purposes, (b) the development of a complex translation interface between the GC scheduler and the cluster batch schedulers could be avoided and (c) the pull allocation model of tasks in XtremWeb (and other GC systems) intrinsically handles the load balancing between the clusters, even in complex scenario where the network conditions vary dynamically.

(II) Programming environment developers may believe that the volatility of LSDS participating nodes mismatches the unavoidable overhead of automatic fault-tolerant techniques making user-based fault tolerance approaches much more relevant. Our results in deploying RPC and MPI applications over XtremWeb using XWRPC and MPICH-V demonstrate that (a) these automatic fault-tolerant programming environments can tolerate a high frequency of faults (as much as more than $1\,\mathrm{min}^{-1}$ in MPICH-V2), (b) the overhead of fault tolerance mechanisms for fault-free executions stays low, and more importantly (c) the performance degradation for faulty executions stays lower than a factor 2 at the highest volatility level.

(III) A third intuitive idea is that providing a high security level for the participating nodes against corruption tentative would impose a high-performance overhead since all of the application actions should be analyzed as potential threats. Our sandbox based on kernel level interposition demonstrates that the interposition cost is negligible and that the execution of a security policy on all system calls of key operations of LSDS systems has negligible impact on performance.

The development of GC systems and the one of XtremWeb have followed a trajectory parallel to the one of Grid systems such as Globus [3] and Unicore [5]. Nevertheless we can observe some convergence elements between GC and Grid. The paper [24] gives many details about the similarities and differences between P2P and Grid systems. From the system developer perspective our experience as the developers of XtremWeb gives a complementary point of view.

Despite a neutral position concerning Grid systems, every architecture evolution we made and we are currently envisioning for our system conducts XtremWeb invariably closer to Grid systems. Of course, the evolution of Globus to GT3 [41] and the notion of Grid services is one reason of this convergence. The service architecture of XtremWeb is the result of an evolution toward a GC system more flexible and secure being able (1) to provide storage and communication sharing in addition to CPU sharing and (2) to propose alternative implementations of functionalities for scheduling and communication between participants. Since Grid services are emerging as a de facto standard for large distributed systems, it is likely that the next generation of XtremWeb will use Grid services as basic building blocks.

At the beginning of XtremWeb we were tempted to design and develop new parallel and distributed programming paradigms more adapted to the system behavior such as scale and volatility than classi-

cal programming environments like RPC and MPI. However, discussions with potential users rapidly convinced us that two factors were against this direction: (1) potential users have already translated their applications several times in the past following the evolution of parallel computing platforms. They were reluctant to spend time on translating their applications to a non-standard and new programming paradigm, (2) they were not convinced of the benefit of GC systems. These factors were the motivations for developing XWRPC and MPICH-V. These two programming environments provide transparent and automatic fault tolerance for applications written in MPI or using the blocking/non-blocking RPC paradigm (Master-Worker, workflow, etc.). RPC and MPI are also two programming paradigms currently available for the Grid. Ultimately, the benefit of this convergence for the users is to keep the number of programming environments to learn very low. As another element of this convergence, the next version of our fault-tolerant MPI implementation (MPICH-V3) will use a hierarchical protocol specifically designed for the Grid being able to harness resources of clusters and GC Systems.

Technical differences between the two systems still exist and are mainly related to their scale and their level of resource control. The technical differences mainly concern (1) deployment (2) fault tolerance and (3) security.

GC systems are concerned by ease of deployment. For harnessing many resources they should provide easy install/uninstall tools and communication protocols automatically dealing with firewall/NAT/proxy issues. As discussed in the paper, GC systems may use cluster nodes as participating resources. This is actually the case for SETI@home for example and XtremWeb. If Grid systems become more general, they would probably extend their resource usage to individual PCs and rely on the same solutions for deployment and communication issues. In this situation, there would be no obvious differences between the resources used by the two systems.

Fault tolerance is mandatory in GC systems since the lack of control allows (1) any participant to leave the systems without any prior mention and (2) distributed and coordinated attacks can be launched against the systems itself. Thanks to their higher level of control of the users and participants, Grid may not be subject of such issues. However, Grid systems should still find solutions to cope with the timely and costly consequences of involuntary faults. More generally, as Grid systems will become larger, faults will become more frequent and the control level on users and participants will become weaker. For this parameter too, we believe that Grid will converge to GC systems with the adoption of strong fault tolerance mechanisms.

Security is certainly the most differentiating factor between the two systems. In Grid, users are the essential source of threat while in GC systems, threats may also come from the application, the infrastructure, the computing nodes and the data (parameters and results). User authentication is implemented carefully in Grid and mostly inexistent in existing GC Systems. Data privacy is enforced in Grid sites by existing well configured mechanisms of the operating systems. Such correct configuration cannot be assumed in GC systems and more importantly the lack of control cannot preclude a participant to spy the data used or produced by an application. Resource protection is implicit in Grid since the users know that their actions are logged and some revocation actions can be decided against them if they are at the origin of a system problem. This is not true in GC systems where the resources computing the application with the parameters and analyzing the results should be self-protected by sandbox. The security of application results is also implicit in Grid because the users trust the institution hosting the resource. In GC system, the lack of trust imposes the use of result certification techniques [43]. Some security systems have been designed like in Ref. [31] and CRISIS [10] for LSDS. Compared to current Grid security designs, they also provide the notion of certificate, delegation, lifetime, revocation, etc., but with more scalability. However, they rely on more complex designs. Thus, it is likely that, because of the lack of control, either GC systems will not provide the trust level of Grid systems or the complexity of a security infrastructure providing the same level of trust would be much higher than the one developed currently for Grid systems.

Based on all the previously discussed parameters, we can clearly conclude a convergence trend between Grid and GC, Grid systems being motivated by more scalability and GC systems by more generality.

References

[1] Web Services. www.webservices.org.
[2] Mithral Communications and Design Inc., The COSM Project, 2002. http://www.mithral.com/projects/cosm/.
[3] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The physiology of the grid: an open grid services architecture for distributed systems integration, Globus Project, 2002, www.globus.org/research/papers/ogsa.pdf.
[4] A.D. Alexandrov, M. Ibel, K.E. Schauser, C.J. Scheiman, SuperWeb: towards a global web-based parallel computing infrastructure, in: Proceedings of the 11th IEEE International Parallel Processing Symposium (IPPS), April 1997.
[5] J. Almond, M. Romberg, The Unicore project: uniform access to supercomputing over the Web, in: Proceedings of the 40th Cray User Group Meeting, Stuttgart, Germany, 1998.
[6] D. Anderson, BOINC: Berkeley Open Infrastructure for Network Computing, 2002.
[7] R. Baldoni, C. Marchetti, Three-tier replication for FT-CORBA infrastructures, Software Pract. Exper. 33 (2003) 767–797.
[8] A. Barak, S. Guday, R.G. Wheeler, The MOSIX distributed operating system: load balancing for UNIX v. 672, Springer-Verlag, New York, NY, 1993, p. 221.
[9] A. Baratloo, M. Karaul, Z. Kedem, P. Wyckoff, Charlotte: metacomputing on the Web, in: Proceedings of the Ninth Conference on Parallel and Distributed Computing Systems, 1996.
[10] E. Belani, A. Vahdat, T. Anderson, M. Dahlin, The CRISIS wide area security architecture, in: Proceedings of the USENIX Security Symposium, San Antonio, Texas, 1998.
[11] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fédak, C. Germain, T. Hérault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Néri, A. Selikhov, MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes, SC02, Baltimore, USA, November 2002.
[12] A. Bouteiller, F. Cappello, T. Hérault, G. Krawezik, P. Lemarinier, F. Magniette, MPICH-V2: A Scalable Fault Tolerant MPI for Volatile Nodes Based on the Pessimistic Sender Based Message Logging, Proceedings of the 16th High Performance Networking and Computing conference (SC'03), Phoenix, USA, November 2003.
[13] T. Brecht, H. Sandhu, M. Shan, J. Talbot, ParaWeb: towards world-wide super-computing, in: Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications, 1996.
[14] H. Casanova, J. Dongarra, NetSolve: a network-enabled server for solving computational science problems, Int. J. Supercomput. Appl. High Perform. Comput., Sage Publications 11 (3) (1997) 212–223.
[15] B.O. Christiansen, P. Cappello, M.F. Ionescu, M.O. Neary, K.E. Schauser, D. Wu, Javelin: Internet-based parallel computing using Java, Concurrency Pract. Exper. 9 (11) (1997) 1139–1160.
[16] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: High-Performance Distributed Computing (HPDC-10), IEEE Press, 2001.
[17] S. Djilali, P2P-RPC: programming scientific applications on peer to peer systems with remote procedure call, in: I. Press (Ed.), Proceedings of the Third International Symposium on Cluster Computing and the Grid, Tokyo, Japan, November 2003.
[18] G. Fagg, J. Dongarra, FT-MPI: fault tolerant MPI, supporting dynamic applications in a dynamic world, in: Euro PVM/MPI User's Group Meeting 2000, Springer-Verlag, Berlin, Germany, 2000, pp. 346–353.
[19] G.E. Fagg, A. Bukovsky, J.J. Dongarra, HARNESS and fault tolerant MPI, Parallel Comput. 27 (11) (2001) 1479–1495.
[20] S. Fanning, Napster: a P2P file sharing, 1999. http://www.napster.com.
[21] G. Fedak, C. Germain, V. Neri, F. Cappello, XtremWeb: a generic global computing platform (CCGRID-2001), in: Special Session Global Computing on Personal Devices, IEEE Press, 2000.
[22] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevcik, P. Wong, Theory and practice in parallel job scheduling, in: D.G. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, Springer-Verlag, 1997, pp. 1–34.
[23] M. Fischer, N. Lynch, M. Paterson, Impossibility of distributed consensus with one faulty process, J. ACM 32 (1985) 374–382.
[24] I. Foster, A. Iamnitchi, On death, taxes, and the convergence of peer-to-peer and grid computing, in: Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS'03), February 2003.
[25] I. Foster, C. Kesselman, J. Nick, S. Tuecke, Grid services for distributed system integration, in: IEEE Comput., June 2002, pp. 37–46.
[26] D.P. Ghormley, D. Petrou, S.H. Rodrigues, A.M. Vahdat, T.E. Anderson, GLUnix: a global layer Unix for a network of workstations, Software Pract. Exper. 28 (9) (1998) 929–961.
[27] J.P. Goux, S. Kulkarni, J. Linderoth, M. Yoder, An enabling framework for master-worker applications on the computational grid, in: I.C. Society (Ed.), Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, Pittsburgh, PA, 2000, pp. 43–50.
[28] W. Gropp, E. Lusk, Fault tolerance in MPI programs, Special Issue of the J. High Perform. Comput. Appl., 2002.
[29] W. Gropp, E. Lusk, N. Doss, A. Skjellum, High-performance, portable implementation of the MPI message passing interface standard, Parallel Comput. 22 (6) (1996) 789–828.
[30] S.M. Inc., RPC: remote procedure call protocol specification version 2, in: Tech. Rep. DARPA-Internet RFC 1057, SUN Microsystems Inc., June 1988.
[31] B. Lampson, M. Abadi, M. Burrows, T. Wobber, Authentication in distributed systems: theory and practice, in: Proceedings of the 13th ACM Symposium on Operating Systems Principles, 13–16 October 1991, pp. 165–182.
[32] M. Litzkow, M. Livny, M. Mutka, Condor—a hunter of idle workstations, in: Proceedings of the Eighth International Conference of Distributed Computing Systems, IEEE Computer Society Press, Madison, Wisconsin, 1988, pp. 104–111.
[33] M.J. Litzkow, M. Livny, M.W. Mutka, Condor—a hunter of idle workstations, in: Proceedings of the Eighth International

Conference on Distributed Computing Systems (ICDCS), IEEE Computer Society, Washington, DC, 1988, pp. 104–111.
[34] C. Mike, M. Pavel, Subterfugue: a frame-work for observing and playing with the reality of software. http://subterfugue.org/.
[35] N. Nisan, S. London, O. Regev, N. Camiel, Globally distributed computation over the internet-the popcorn project, in: Proceedings for the 18th International Conference on Distributed Computing Systems, 1998.
[36] CORBA 2.1 Common Object Request Broker: Architecture and Specification Revision 2.0, July 1995, updated July 1996, http://www.opengroup.org/openbrand/register/orm0.htm.
[37] H. Pedroso, L.M. Silva, J.G. Silva, Web-based metacomputing with JET, in: Proceedings of the ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation, ACM, June 1997.
[38] J.S. Plank, H. Casanova, M. Beck, J. Dongarra, Deploying fault tolerance and task migration with NetSolve, Future Gener. Comput. Syst. 15 (1999) 745–755.
[39] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A Scalable Content Addressable Network, Technical Report TR-00-010, Berkeley, CA, 2000.
[40] A. Rowstron, P. Druschel, Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems, Lecture Notes in Computer Science, 2001, p. 2218.
[41] T. Sandholm, J. Gawor, Globus Toolkit 3 Core—a grid service container FrameWork, in: Globus Toolkit Core White Paper, July 2003. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf.
[42] L.F.G. Sarmenta, Volunteer computing, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, USA, June 2001.
[43] L.F.G. Sarmenta, Sabotage-tolerance mechanisms for volunteer computing systems, in: Proceedings of the ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01), Brisbane, Australia, 15–18 May 2001.
[44] L.F.G. Sarmenta, S. Hirano, S.A. Ward, Towards Bayanihan: builing an extensible framework for Volonteer Computing using Java, in: Proceedings of the ACM Workshop on Java for High-Performance Network Computing, 1998.
[45] S. Saroiu, P.K. Gummadi, S.D. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking 2002 (MMCN'02), San Jose, CA, January 2002. http://www.cite-seer.nj.nec.com/saroiu02measurement.html.
[46] M. Sato, M. Hirano, Y. Tanaka, S. Sekiguchi, OmniRPC: a grid RPC facility for cluster and global computing in OpenMP, in: Springer (Ed.), Proceedings of the Workshop on OpenMP Applications and Tools 2001, volume LNCS 2104, West Lafayette, IN, July 2001, pp. 130–135.
[47] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, H. Takagi, Ninf: a network based information library for global world-wide computing infrastructure, in: Proceedings of the High-Performance Computing and Networking, International Conference and Exhibition, HPCN Europe, volume LNCS 1225, Vienna, Austria, Springer, April 1997, pp. 491–502.
[48] S.J. Sciutto, Aires: Air Showers Extended Simulation, Department of Physics of the Universidad Nacional de La Plata, Argentina, 1995. http://www.fisica.unlp.edu.ar/auger/aires/.
[49] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, H. Casanova, GridRPC: a remote procedure call API for grid computing, in: Technical Report, University of Tennesse, ICL-UT-02-06, June 2002.
[50] J. Shoch, J. Hupp, Computing practices: the 'Worm' programs—early experience with a distributed computation, Comm. ACM 25 (3) (1982) 172–180.
[51] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI: The Complete Reference, The MIT Press, 1996. http://www.netlib.org/utk/papers/mpi-book/mpi-book.html.
[52] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, Technical Report TR-819, MIT, 2001.
[53] B.Y. Zhao, J.D. Kubiatowicz, A.D. Joseph, Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

Franck Cappello holds a Research Director position at INRIA, after having spent 8 years as CNRS researcher. He leads the Grand-Large project at INRIA and the Cluster and Grid group at LRI. He has authored more than 50 papers in the domains of High Performance Programming, Desktop Grids and Fault-tolerant MPI. He is editorial board member of the "International Journal on GRID Computing" and steering committee member of IEEE/ACM CCGRID. He organizes annually the Global and Peer-to-Peer Computing workshop. He leads the XtremWeb (Desktop Grid) and MPICH-V (Fault-tolerant MPI) projects. He is currently involved in two new projects: Grid eXplorer (a Grid Emulator) and Grid'5000 (a Nation Wide Experimental Grid Testbed).

Samir Djilali is a PhD candidate in computer science from Paris South University (France). He received his MS in computer science from Paris South University in 2001. He is a member of the Grand-Large INRIA project. His work focuses on programming models and fault tolerance for large-scale distribute systems.

Gilles Fedak is a Post-Doctoral fellow in the GRAIL Computer Science Laboratory at the University of California at San Diego. His research interests include Grid and Global Computing environments. He was the main contributor to the XtremWeb project. He received his PhD in computer science from Paris South University, France, in 2003.

Thomas Herault is an assistant professor at the Paris South University (France). He defended its PhD on the mending of transient failure in self-stabilizing systems in 2003, under the supervision of Professor Joffroy Beauquier. He is a member of the Grand-Large INRIA project and works on fault-tolerant protocols in distributed systems. He contributes to the MPICH-V project of fault-tolerant MPI and to fault tolerance for XtremWeb.

Frédéric Magniette received his PhD from Paris South University, France, in 2003. He is currently a CNRS engineer. His research interests include self-stabilizing systems and security in large-scale distributed systems.

Vincent Néri is a CNRS engineer at LRI (Computer Science Laboratory) at Paris South University. His research interests include large-scale distributed systems. He is a major contributor of XtremWeb project. He is a member of the Grand-Large INRIA project. He received his PhD in computer science from Paris South University in 1995.

Oleg Lodygensky is an engineer in Computing Science at LAL (a High Energy Physics Laboratory of the CNRS IN2P3 Institute), Orsay, France, since 1994. .He is also a PhD student (since 2001) at LRI, Orsay, France, under the direction of Franck Cappello. The aim of this thesis is to evaluate the possibility of Desktop Grid use among the High Energy Physics Community, a highly involved community in Grid (and in DataGrid, more specifically).