INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Grand-Large

Calcul parallèle et distribué à grande échelle

Futurs

THEME 1A

Activity Report

2003

Table of contents

1. Team

Head of project-team
Franck Cappello [Research Director at INRIA-Futurs]

Topic leaders
Franck Cappello [Middleware Design, Implementation and Test]
Joffroy Beauquier [Verification]
Serge Petiton [Large Scale Numerical Computing]

Administrative assistant
Gina Grisvard

Permanent Members
Joffroy Beauquier [Professor at University of Paris-Sud]
Franck Cappello [Research Director at INRIA-Futurs]
Serge Petiton [Professor at University of Science and Technology of Lille]
Brigitte Rozoy [Professor at University of Paris-Sud]
Sebastien Tixeuil [Assistant Professor at Paris Sud University]

Non Permanent Position
Thomas Herault [Teaching Assistant at Paris-Sub University]

Ph. D. student
Lamine Aouad [ACI GRID at Lille 1 University]
Georges Boscilca [MESR Grant (LRI)]
Aurelien Bouteiller [MESR Grant (LRI)]
Samir Djilali [MESR Grant (LRI)]
Philippe Gauron [MESR Grant (LRI)]
Benoit Hudzia [Franco-Irish Grant (LIFL)]
Geraud Krawezik [CIFRE Grant (LRI/EADS)]
Pierre Lemarinier [MESR Grant (LRI)]
Oleg Lodygensky [LaL Engineer (Laboratoire de l'Accelerateur Lineaire)]

Research scientist (partner)
Pierre Fraigniaud [Research Director at CNRS]
Laurent Rosaz [Assistant Professor at Paris Sud University]

Project technical staff
Tangui Morlier [INRIA Associate Engineer]
Vincent Neri [CNRS Study Engineer]

2. Overall Objectives

2.1. Grand-Large General Objectives

Grand-Large is a Grid research project investigating the issues raised by computing on Large Scale Distributed Systems (LSDS), where participants execute different applications on shared resources belonging to other participants, possibly geographically and administratively independent. More specifically, we consider large scale parallel and distributed computing on P2P, Global Computing and Desktop Grid systems. Our research focuses on middleware and low level programming environments design, proof and experiments. Fundamentally, we address the impact of LSDS, gathering several methodological tools: theoretical models, simulators, emulators and real size systems.

The project aims:

1.

to study experimentally, and formally, the fundamental mechanisms of LSDS for high performance computing;

2. to design, implement, validate and test real software, middleware and platform;
3. to define, evaluate and experiment approaches for programming applications on these platforms.

Compared to other European and French projects, we gather skills in large scale systems (large scale scheduling, volatility tolerance, heterogeneity, inter administration domain security, etc.) acquired with the XtremWeb project (LRI, Cluster and Grid team), formal design and validation of algorithms and protocols for distributed systems (LRI, Parallelism team) and programming, evaluation, analysis and definition of programming languages and environments for parallel architectures and distributed systems (LIFL, methodologies and parallel algorithms).

This project pursues short and long term researches aiming to have scientific and industrial impacts. Research topics include:

1. the design of a middleware enlarging the application domain of Desktop Grid;
2. resource discovery engine on large scale system with volatil participants;
3. large scale storage on volatile nodes;
4. simulation of large scale scheduling;
5. fault tolerant MPI for large scale systems;
6. algorithm for large scale fault tolerance;
7. protocol verification;
8. algorithms, programming and evaluation of scientific applications on desktop Grids;
9. tools and languages for large scale computing.

These researches should have some applications in the domain of LSDS, Grid and large clusters.

At a longer term, we investigate the convergence conditions of Global Computing, P2P and Grid systems (how Grid Services can be used in Desktop Grid) and experimental tools for improving the methodology associated with research in LSDS. For example we have the responsibility of the Grid eXplorer project founded by the French ministry of research and we are deeply involved in the Grid5000 project.

3. Scientific Foundations

3.1. Large Scale Distributed Systems (LSDS)

What makes a fundamental difference between pioneer Global Computing 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 features that has to be taken into account if the system should scale to a very 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 reconnect the system in the same way. This feature raises many novel issues: under such assumptions, the system may 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 [29] no consensus could be achieved on such a system. Another example of feature is the complete lack of control of nodes and networks. 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 with the fact that any node may intermittently injects Byzantine faults. These features set up a new research context in distributed systems. The Grand-Large project aims at investigating theoretically as well as experimentally the fundamental mechanisms of LSDS, especially for the high performance computing applications.

3.1.1. Computing on Large Scale Global Computing systems

Currently, largest LSDS are used for Computing (SETI@home, Folding@home, Decrypthon, etc.), file exchanges (Napster, Kazaa, eDonkey, Gnutella, etc.), networking experiments (PlanetLab, Porivo) and communication such as instant messaging and phone over IP (Jabber, Skype). In the High Performance Computing domain, LSDS have emerged while the community was considering clustering and hierarchical designs as good performance-cost tread-offs.

LSDS as a class of Grid systems, essentially extends the notion of computing beyond the frontier of administration domains. The very first paper discussing this type of systems [30] presented the Worm programs and several key ideas that are currently investigated in autonomous computing (self replication, migration, distributed coordination, etc.). LSDS inherit the principle of aggregating inexpensive, often already in place, resources, from past research in cycle stealing/resource sharing. Due to its high attractiveness, cycle stealing has been studied in many research projects like Condor [31], Glunix [32] and Mosix [33], to cite a few. A first approach to cross administration domains was proposed by Web Computing projects such as Jet [38], Charlotte [39], Javeline [40], Bayanihan [34], SuperWeb [35], ParaWeb [36] and PopCorn [37]. These projects have emerged with Java taking benefit of the virtual machine properties: high portability across heterogeneous hardware and OS, 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 Global Computing systems like COSM [41], BOINC [42] and XtremWeb [43].

The high performance potential of LSDS platforms has also raised a significant interest in the industry. Companies like Entropia [44], United Devices [45], Platform [46], Grid systems [47] and Datasynapse [48] propose LSDS 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.

LSDS systems share with Grid a common objective: to extend the size and accessibility of a computing infrastructure beyond the limit of a single administration domain. In [49], 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 perspective, we consider 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.

3.1.2. Building a Large Scale Distributed System for Computing

This set of studies considers the XtremWeb project as the basis for research, development and experimentation. This LSDS middleware is already operational. This set gathers 4 studies aiming at improving the mechanisms and enlarging the functionalities of LSDS dedicated to computing. The first study considers the architecture of the resource discovery engine which, in principle, is close to an indexing system. The second study concerns the storage and movements of data between the participants of a LSDS. In the third study, we will address the issue of scheduling in LSDS in the context of multiple users and applications. Finally the last study seeks to improve the performance and reduce the resource cost of the MPICH-V fault tolerant MPI for desktop grids.

3.1.2.1. The resource discovery engine

A multi-users/multi-applications LSDS system for computing would be in principle very close to a P2P file sharing system such as Napster [50], Gnutella [50] and Kazaa [51], except that the ultimate shared resource is the CPUs instead of files. The scale and lack of control are common features 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. As an example, recent P2P projects have proposed distributed indexing systems like CAN [52], CHORD[53], PASTRY [54] and TAPESTRY [55] that could be used for resource discovery in a LSDS dedicated to computing.

The resource discovery engine is composed of a publishing system and a discovery engine, which allow a client of the system to discover the participating nodes offering some desired services. Currently, there is as much resource discovery architectures as LSDS and P2P systems. The architecture of a resource discovery engine is derived from some expected features such as speed of research, speed or reconfiguration, volatility tolerance, anonymity, limited used of the network, matching between the topologies of the underlying network and the virtual overlay network. The currently proposed architectures are not well motivated and seem to be derived from arbitrary choices.

This study has two objectives: a) compare some existing resource discovery architectures (centralized, hierarchical, fully distributed) with relevant metrics; and b) potentially propose a new protocol improving some parameters. Comparison will consider the theoretical aspects of the resource discovery engines as well as their actual performance when exposed to real experimental conditions.

3.1.2.2. Data storage and movement

Application data movements and storage are major issues of LSDS since a large class of computing applications requires the access of large data sets as input parameters, intermediary results or output results.

Several architectures exist for application parameters and results communication between the client node and the computing ones. XtremWeb uses an indirect transfer through the task scheduler which is implemented by a middle tier between client and computing nodes. When a client submits a task, it encompasses the application parameters in the task request message. When a computing node terminates a task, it transfers it to the middle tier. The client can then collect the task results from the middle tier. BOINC [42] follows a different architecture using a data server as intermediary node between the client and the computing nodes. All data transfers still pass through a middle tier (the data server). DataSynapse [48] allows direct communications between the client and computing nodes. This architecture is close to the one of file sharing P2P systems. The client uploads the parameters to the selected computing nodes which return the task results using the same channel. Ultimately, the system should be able to select the appropriate transfer approach according to the performance and fault tolerance issues. We will use real deployments of XtremWeb to compare the merits of these approaches.

Currently there is no LSDS system dedicated to computing that allows the persistent storage of data in the participating nodes. Several LSDS systems dedicated to data storage are emerging such as OCEAN Store [56] and Ocean [83]. Storing large data sets on volatile nodes requires replication techniques. In CAN and

Freenet, the documents are stored in a single piece. In OceanStore, Fastrack and eDonkey, the participants store segments of documents. This allows segment replications and the simultaneous transfer of several documents segments. In the CGP2P project, a storage system called US has been proposed. It relies on the notion of blocs (well known in hard disc drivers). Redundancy techniques complement the mechanisms and provide raid like properties for fault tolerance. We will evaluate the different proposed approaches and the how replication, affinity, cache and persistence influence the performances of computational demanding applications.

3.1.2.3. Scheduling in large scale systems

Scheduling is one of the system fundamental mechanisms. Several studies have been conducted in the context of Grid mostly considering bag of tasks, parameter sweep or workflow applications [57], [58]. Recently some researches consider scheduling and migrating MPI applications on Grid [60]. Other related researches concern scheduling for cycle stealing environments [59]. Some of these studies consider not only the dynamic CPU workload but also the network occupation and performance as basis for scheduling decisions. They often refere to NWS which is a fundamental component for discovering the dynamic parameters of a Grid.

There are very few researches in the context of LSDS and no existing practical ways to measure the workload dynamics of each component of the system (NWS is not scalable). There are several strategies to deal with large scale system: introducing hierarchy or/and giving more autonomy to the nodes of the distributed system.

The purpose of this research is to evaluate the benefit of these two strategies in the context of LSDS where nodes are volatile. In particular we are studying algorithms for fully distributed and asynchronous scheduling, where nodes take scheduling decisions only based on local parameters and information coming from their direct neighbours in the system topology.

In order to understand the phenomena related to full distribution, asynchrony and volatility, we are building a simulation framework called SimLargeGrid. This framework, based on the Swarm [72] multi-agent simulator, allows describing an algorithm, simulating its execution by thousands of nodes and visualizing dynamically the evolution of parameters, the distribution of tasks among the nodes in a 2D representation and the dynamics of the system with a 3D representation. We beleive that visualization and experimentation are a first necessary setp before any formalization since we first need to understand the fundamental caracteristics of the systems before beeing able to modelize them.

3.1.2.4. Extension of MPICH-V

MPICH-V is a research effort with theoretical studies, experimental evaluations and pragmatic implementations aiming to provide a MPI implementation based on MPICH [61], featuring multiple fault tolerant protocols.

There is a long history of research in fault tolerance for distributed systems. We can distinguish the automatic/transparent approach from the manual/user controlled approach. The first approach relies either on coordinated checkpointing (global snapshot) or uncoordinated checkpointing associated with message logging. A well known algorithm for the first approach has been proposed by Chandy and Lamport [62]. This algorithm requires restarting all processes even if only one process crashes. So it is believed not to scale well. Several strategies have been proposed for message logging: optimistic [63], pessimistic [64], causal [65]. Several optimizations have been studied for the three strategies. The general context of our study is high performance computing on large platforms. One of the most used programming environments for such platforms is MPI.

Whithin the MPICH-V project, we have developed and published 3 original fault tolerant protocols for MPI: MPICH-V1 [66], MPICH-V2 [67], MPICH-V/CL [68]. The two first protocols rely on uncoordinated checkpointing associated with either remote pessimistic message logging or sender based pessimistic message logging. We have demonstrated that MPICH-V2 outperforms MPICH-V1. MPICH-V/CL implements a coordinated checkpoint strategy (Chandy-Lamport) removing the need of message logging. MPICH-V2 and V/CL are concurrent protocols for large clusters. We have compared them considering a new parameter for evaluating the merits of fault tolerant protocols: the impact of the fault frequency on the performance. We have demonstrated that the stress of the checkpoint server is the fundamental source of performance differences between the two techniques. Under the considered experimental conditions, message logging becomes more relevant than coordinated checkpoint when the fault frequency reach 1 fault every 4 hours, for a cluster of 100 nodes sharing a single checkpoint server, considering a data set of 1 GB on each node and a 100 Mb/s network.

The next step in our research is to investigate a protocol dedicated for hierarchical desktop Grid (it would also apply for Grids). In such context, several MPI executions take place on different clusters possibly using heterogeneous networks. An automatic fault tolerant MPI for HDG or Grids should tolerate faults inside clusters and the crash or disconnection of a full cluster. We are currently considering a hierarchical fault tolerant protocol combined with a specific runtime allowing the migration of full MPI executions on clusters independently of their high performance network hardware.

The performance and volatility tolerance of MPICH-V make it attractive for :

1. large clusters;
2. clusters made from collection of nodes in a LAN environment (Desktop Grid);
3. Grid deployments harnessing several clusters;
4. and campus/industry wide desktop Grids with volatile nodes (i.e. all infrastructures featuring synchronous networks or controllable area networks).

3.2. Volatility and Reliability Processing

In a global computing application, users voluntarily lend the machines, during the period they dont use them. When they want to reuse the machines, it is essential to give them back immediately. There is no time for saving the state of the computation. Because the computer may not be available again, it is necessary to organize checkpoints. When the owner takes control of his machine, one must be able to continue the computation on another computer from a checkpoint as near as possible from the interrupted state. The problem that arises from this way of managing computations are numerous and difficult. They can be put into two categories: synchronization and repartition problems.

- Synchronization problems (example).
  Suppose that the machine that is supposed to continue the computation is fixed and has a recent checkpoint. It would be easy to consider that this local checkpoint is a component of a global checkpoint and to simply rerun the computation. But on one hand the scalability and on the other hand the frequency of disconnections makes the use of a global checkpoint totally unrealistic. Then the checkpoints have to be local and the problem of synchronizing the recovery machine with the application is raised.
- Repartition problems (example).
  As it is also unrealistic to wait for the computer to be available again before rerunning the interrupted application. One has to design a virtual machine organization, where a single virtual machine is implemented as several real ones. With too few real machines for a virtual one, one can produce starvation; with too many, the efficiency is not optimal. The good solution is certainly in a dynamic organization.

These types of problems are not new ([86]). They have been studied deeply and many algorithmic solutions and implementations are available. What is new here and makes these old solutions not usable is scalability. Any solution involving centralization is impossible to use in practice. Previous works validated on former networks can not be reused.

3.2.1. Reliability processing

We voluntarily presented in a separate section the volatility problem because its specific both with respect to type of failures and to frequency of failures. But in a general manner, as any distributed system, a global computing system has to resist to a large set of failures, from crash failures to Byzantine failures, that are related to incorrect software or even malicious actions (unfortunately, this hypothesis has to be considered as shown by DECRYPTON project or the use of erroneous clients in SETI@HOME project), with transient failures as loss of message duplication in between. On the other hand, failures related accidental or malicious memory corruptions have to be considered because they are directly related of the very nature of the Internet. Traditionally, two approaches (masking and non-masking) have been used to deal with reliability problems. A masking solution hides the failures to the user, while a non-masking one may let the user notice that failures occur. Here again, there exists a large literature on the subject (cf. [84] [87] [85] for surveys). Masking techniques, generally based on upon consensus, because they systematically use generalized broadcasting are not scalable. The self-stabilizing approach (a non-masking solution) is well adapted (specifically its time adaptive version, cf. [91] [92], [88], [89], [90]) for three main reasons:

1. Low overhead when stabilized. Once the system is stabilized, the overhead for maintaining correction is slow because it only involves communications between neighbors.
2. Good adaptivity to the reliability level. Except when considering a system that is continuously under attacks, self-stabilization provides very satisfying solutions. The fact that during the stabilization phase, the correctness of the system is not necessarily satisfied is not a problem for all kind of application.
3. Lack of global administration of the system. A peer to peer system does not admit a centralized administrator that would be recognized by all components. A human intervention is thus not feasible and the system has to recover by itself from the failures of one or several components, that is precisely the feature of self-stabilizing systems.

We propose:

1. To study the reliability problems arising from a global computing system, and to design self-stabilizing solutions, with a special care for the overhead.
2. For problem that can be solved despite continuously unreliable environment (such as information retrieval in a network), to propose solutions that minimize the overhead in space and time resulting from the failures when they involve few components of the system.
3. For most critical modules, to study the possibility to use consensus based methods.
4. To build an adequate model for dealing with the tradeoff between reliability and cost.

3.2.2. Verification of protocols

For the past few years, a number of distributed algorithms or protocols that were published in the best conferences or scholar journals were found to be incorrect afterwards. Some have been exploited for several years, appearing to behave correctly. We do not pretend to design and implement fault free and vulnerability free systems, but we want at least to limit their failures. This goal is achieved by the formal verification, at an abstract level, of the implemented solutions. Obviously, algorithms are not to be verified by hand (incorrect algorithms were provided with proofs), but rather by verification tools we developed (MARELLA) or proof assistants. We propose that a substantial effort is done towards modelization and verification of probabilistic protocols, which offer in a large number of cases efficient and low cost solutions. We also propose to design a model that includes the environment. Indeed, computations of a distributed system are non-deterministic due to the influence of numerous external factors, such as the communication delays due to traffic overhead, the fact that failures can occur somewhere rather than somewhere else, etc. To prove a protocol independently of its environment is pointless, and this is why the environment must be part of the model.

3.3. Parallel Programming on Peer-to-Peer Platforms (P5)

Scientific applications that have traditionally performed on supercomputers may now run on a variety of heterogeneous resources geographically distributed. New grand challenge applications would have to be solved on large scale P2P systems. Peer-to-Peer computing paradigm for large scale scientific and engineering applications is emerging as a new potential solution for end-user scientist and engineers. We have to experiment and to evaluate such programming to be able to propose the larger possible virtualisation of the underlying complexity for the end-user.

3.3.1. Large Scale Computational Sciences and Engineering

Parallel and distributed scientific application developments and resource managements in these environments are a new and complex undertaking. In scientific computation, the validity of calculations, the numerical stability, the choices of methods and software are depending of properties of each peer and its software and hardware environments; which are known only at run time and are indeterminists. The research to obtain acceptable frameworks, methodologies, languages and tools to allow end-users to solve accurately their applications in this context is capital for the future of this programming paradigm.

GRID scientific and engineering computing exists already since a decade. Since the last few years, the scale of the problem sizes and the global complexity of the applications increase rapidly [78]. The scientific simulation approach is now general in many scientific domains, in addition to theoretical and experimental aspects, often link to more classic methods. Several applications would be computed on world-spread networks of heterogeneous computers using some web-based Application Server Provider (ASP) dedicated to targeted scientific domains. New very strategic domains, such as Nanotechnologies, are in the forefront of these applications. The development in this very important domain and the leadership in many scientific domains will depend in a close future to the ability to experiment very large scale simulation on adequate systems [80], [81]. The P2P scientific programming is a potential solution, which is based on existing computers and networks. The present scientific applications on such systems are only concerning problems which are mainly data independents: i.e. each peer does not communicate with the others. To come at his age, P2P programming has to be able to develop parallel programming with more sophisticate dependencies between peers. It is the goal of our researches.

3.3.2. Experimentations and Evaluations

We have, first, to experiment on large P2P platforms to be able to obtain a realistic evaluation of the performance we can expect. We can also set some hypothesis on peers, networks, and scheduling to be able to have theoretical evaluations of the potential performance. We follow these two tracks. We choose a classical linear algebra method well-adapted to large granularity parallelism and asynchronous scheduling: the block Gauss-Jordan method to invert dense very large matrices. We also choose the calculation of one matrix polynomial, which generate computation schemes similar to many linear algebra iterative methods, well-adapted for very large sparse matrices. Thus, we were able to theoretically evaluate the potential throughput with respect to several parameters such as the matrix size and the multicast network speed. Since these evaluations, we begin to experiment the same parallel methods on a few dozen peer XtremWeb P2P Platform. We plan to continue these experimentations on larger platforms to compare these results to the theoretical ones. Then, we would be able to extrapolate and obtain potential performance for some scientific applications. Experimentations and evaluation for several linear algebra methods for large matrices on P2P systems will always be developed all along the Grand Large project, to be able to confront the different results to the reality of the existing platforms. As a challenge, we would like to efficiently invert a dense matrix of size one million using a several thousand peer platform.

Beyond the experimentations and the evaluations, we propose the basis of a methodology to efficiently program such platforms, which allow us to define languages, tools and interface for the end-user.

3.3.3. Languages, tools and Interface

The underlying complexity of the Large Scale P2P programming has to be mainly virtualized for the end-user. We have to propose an interface between the end-user and the middleware which may extract the end-user expertise or propose an on-the-shelf general solution. Targeted applications concern very large scientific problems which have to be developed using component technologies and up-to-dated software technologies.

We may develop component-based technology interface which express the dependencies between computing tasks which composed the parallel applications. Then, instead of computing task we will manage components. We introduced the YML language which allows us to express the dependencies between components, specified using XML. Nevertheless, many component criteria depend of peer characteristics and are known only at runtime. Then, we had to introduce different classes of components, depending of the level of abstraction they are concern to. A component catalogue has to be at the end-user level and another one has to be at the middleware and peer level. Then, a scheduler has to attribute a computing component to a peer with respect to the software proposed by this one, or has to decide to load new software to the targeted peer.

The YML framework and language propose a solution to develop scientific applications to P2P platform. An end-user can directly develop programs using this framework. Nevertheless, many end-users would prefer to do not program at this component and dependency graph level. Then, an interface has to be proposed, using the YML framework. This interface may be dedicated to a special scientific domain to be able to focus on the end-user vocabulary and P2P programming knowledge.

Based on the SPIN project, we plan to develop such version based on the YML framework and language. The first targeted scientific domain will be very large linear algebra for dense or sparse matrices.

3.4. Methodology and technologies for Large Scale Distributed Systems

Research in the context of LSDS involves understanding large scale phenomena from the theoretical point of view up to the experimental one under real life conditions. The general research context should also considers the fundamental technological trend toward a convergence between Grid and P2P systems.

3.4.1. Metodology

One key aspects of the impact of large scale on LSDS is the emergence of phenomena witch are not coordinated, intended or expected. These phenomena are the results of the combination of static and dynamic features of each component of LSDS: nodes (hardware, OS, workload, volatility), network (topology, congestion, fault), applications (algorithm, parameters, errors), users (behavior, number, friendly/aggressive).

Grand-Large aims at gathering several complementary techniques to study the impact of large scale in LSDS: theoretical models, simulation, emulation and experimentation on real platforms. Fundamental aspects of LSDS as well as the development of middleware platforms are already existing in Grand-Large. We are also involved in the development and deployment of simulators and emulators and real platforms (testbed).

We are currently developing a simulator of LSDS called SimLargeGrid aiming at discovering, understanding and managing implicit uncoordinated large scale phenomena. Several Grid simulators have been developed by other teams: SimGrid [69] GridSim [70], Briks [71]. All these simulators considers relatively small scale Grids. They have not been designed to scale and simulate 10 K to 100 K nodes. Other simulators have been designed for large multi-agents systems such as Swarm [72] but many of them considers synchronous systems where the system evolution is guided by phases. SimLargeGrid is built from Swarm and adds asynchrony in the simulator, node volatility and a set of specialized features for controlling and measuring the simulation of LSDS. To exemplify the need of such simulator, we are first considering the fully distributed scheduling problem. Using SimLargeGrid for comparing several algorithms, we have already demonstrate the need for complementary visualization tools, showing the evolution of key system parameters, presenting the distributed system topology, nodes and network global trends in a 2 dimensional shape and presenting the dynamics of the system component activity in a 3 dimensional shape. Using this last representation, we have discover unexpected large scale phenomena which would be very difficult to predict by a theoretical analysis of the simulated platform features and the scheduling algorithms.

Emulation is another tool for experimenting systems and networks with a higher degree of realism. Compared to simulation, emulation can be used to study systems or networks 1 or 2 orders of magnitude smaller in terms of number of components. However, emulation runs the actual OS/middleware/applications on actual platform. Compared to real testbed, emulation considers conducting the experiments on a fully controlled platform where all static and dynamic parameters can be controlled and managed precisely. Another advantage of emulation over real testbed is the capacity to reproduce experimental conditions. Several implementations/configurations of the system components can be compared fairly by evaluating them under the similar static and dynamic conditions. Grand-Large is leading one of the largest Emulator project in Europe called Grid explorer. This project uses a 1K CPUs cluster as hardware platform and gathers 24 experiments of 80 researchers belonging to 13 different laboratories. Experiments concern developing the emulator itself and use of the emulator to explore LSDS issues. (www.lri.fr/~fci/GdX/)

Grand-Large members are also involved in the French Grid 5000 project which intents to deploy an experimental Grid testbed for computer scientists. This testbed may feature up to 5000 K CPUs gathering the resources of about 10 clusters geographically distributed over France. The clusters will be connected by a high speed network (Renater or/and other). Grand-Large is a leading team in Grid 5000, chairing the eGrid 5000 Specific Action of the CNRS which is intended to prepare the deployment and installation of Grid 5000. eGrid 5000 gathers about 30 engineers, researchers and team directors who have frequent meetings, discussing about the testbed security infrastructure, experiment setup, cluster coordination, experimental result storage, etc. (www.lri.fr/~fci/AS1/)

3.4.2. Technological trends

The development of LSDS has followed a trajectory parallel to the one of Grid systems such as Globus [73] and Unicore [74]. Nevertheless we can observe some convergence elements between LSDS and Grid. The paper [49] gives many details about the similarities and differences between P2P and Grid systems. From the technological perspective, the evolution of Globus to GT3 [75] with the notion of Grid services is one reason of this convergence. The evolution of LSDS toward more generic and secure systems being able to provide CPU, storage and communication sharing among participants is another element of this convergence, since the notion of controllable services is likely to emerge from this perspective of more generality and flexibility.

Nowadays, Grid Computing is considering the notion of services through OGCSA [76] and OGSI [77]. A Grid 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...) of Web Services [82]. A typical LSDS platform gathering client nodes submitting requests to a coordination service which schedules them on a set of participating nodes can be implemented in term of services: the coordination 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 coordination service; the worker service computes tasks and sends their results back to the coordination service. Note that the implementation of the coordination 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.

Thus we believe that LSDS could benefit from the standardization effort conducted in the Grid context by reusing the same concepts of services and by adopting the same standards (OGSA and OGSI). For example, the next version of XtremWeb will be implemented by a set of Grid services.

4. Application Domains

4.1. Building a Large Scale Distributed System for Computing

The main application domain of the Large Scale Distributed System developed in Grand-Large is high performance computing. The two main programming models associated with our platform (RPC and MPI) allow to program a large variety of distributed/parallel algorithms following computational paradigms like bag of tasks, parameter sweep, workflow, dataflow, master worker, recursive exploration with RPC, and SPMD with MPI. The RPC programming model can be used to execute concurrently different applications codes, the same application code with different parameters and library function codes. In all these cases, there is no need to change the code. The code must only be compiled for the target execution environment. LSDS are particularly useful for users having large computational needs. They could typically be used in Research and Development departments of Pharmacology, Aerospace, Automotive, Electronics, Petroleum, Energy, Meteorology industries. LSDS can also be used for other purposes than CPU intensive applications. Other resources of the connected PCs can be used like their memory, disc space and networking capacities. A Large Scale Distributed System like XtremWeb can typically be used to harness and coordinated the usage of these resources. In that case XtremWeb deploys on Workers services dedicated to provide and manage a disc space and the network connection. The storage service can be used for large scale distributed fault tolerant storage and distributed storage of very large files. The networking service can be used for server tests in real life conditions (workers deployed on Internet are coordinated to stress a web server) and for networking infrastructure tests in real like conditions (workers of known characteristics are coordinated to stress the network infrastructure between them).

4.2. Security and reliability of network control protocols

The main application domain for self-stabilizing and secure algorithms is LSDS where correct behaviours must be recovered within finite time. Typically, in a LSDS (such as a high performance computing system), a protocol is used to control the system, submit requests, retrieve results, and ensure that calculus is carried out accordingly to its specification. Yet, since the scale of the system is large, it is likely that nodes fail while the application is executing. While nodes that actually perform the calculus can fail unpredictably, a self-stabilizing and secure control protocol ensures that a user submitting a request will obtain the corresponding result within (presumably small) finite time. Examples of LSDS where self-stabilizing and secure algorithms are used, include global computing platforms, or peer to peer file sharing systems.

Another application domain is routing protocols, which are used to carry out information between nodes that are not directly connected. Routing should be understood here in its most general acceptance, e.g. at the network level (Internet routing) or at the application level (on virtual topologies that are built on top of regular topologies in peer to peer systems). Since the topology (actual or virtual) evolves quickly through time, self-stabilization ensures that the routing protocol eventually provides accurate information. However, for the protocol to be useful, it is necessary that it provides extra guarantees either on the stabilization time (to recover quickly from failures) or on the routing time of messages sent when many faults occur.

Finally, additional applications can be found in distributed systems that are composed of many autonomous agents that are able to communicate only to a limited set of nodes (due to geographical or power consumption constraints), and whose environment is evolving rapidly. Examples of such systems are wireless sensor networks (that are typically large of 10000+ nodes), mobile autonomous robots, etc. It is completely unrealistic to use centralized control on such networks because they are intrinsically distributed; still strong coordination is required to provide efficient use of resources (bandwidth, battery, etc.).

4.3. End-User Tools for Computational Science and Engineering

Another Grand Large application domain is Large Scale Programming for Computational Science and Engineering. Two main approaches are proposed. First, we have to experiment and evaluate such programming. Second, we have to develop tools for end-users.

In addition to the classical supercomputing and the GRID computing based on virtual organization, the large scale P2P approach proposes new computing facilities for computational scientists and engineers. Thus, on one hand, it exists many applications, some of them are classical, such as Computational Fluid Dynamic or Quantum Physic ones, for example, and others are news and very strategic such as Nanotechnologies, which will have to use a lot of computing power for long period of time in the close future. On another hand, it emerges a new large scale programming paradigm for existing computers which can be accessible by scientific and engineer end-users for all classical application domains but also by new ones, such as some Non-Governmental Organisations. During a first period, many applications would be based on large simulations rather than classical implicit numerical methods, which are more difficult to adapt for such large problems and new programming paradigm. Nevertheless, we expected that more complex implicit methods would be adapted in the future for such programming. The potential number of peer and the planed evolution of network communications, especially multicast ones, would permit to contribute to solve some of the larger grand challenge scientific applications.

Simulations and large implicit methods would always have to compute linear algebra routines, which will be our first targeted numerical methods (we also remark that the powerful worldwide computing facilities are still rated using a linear algebra benchmark [www.top500.org]). We will especially first focus on divide-and-conquer and block-based matrix methods to solve dense problems and on iterative hybrid methods to solve sparse matrix problems. As these applications are utilized for many applications, it is possible to extrapolate the results to different scientific domains.

Many smart tools have to be developed to help the end-user to program such environments, using up-to-date component technologies and languages. At the actual present stage of maturity of this programming paradigm for scientific applications, the main goal is to experiment on large platforms, to evaluate and extrapolate performance, and to propose tools for the end-users; with respect to many parameters and under some specify hypothesis concerning scheduling strategies [18] and multicast speeds [79]. We have to always replace the end-user at the center of this scientific programming. Then, we have to propose a framework to program P2P architectures which completely virtualized the P2P middleware and the heterogeneous hardware. Our approach is based, on one hand, on component programming and coordination languages, and on one another hand, to the development of an ASP, which may be dedicated to a targeted scientific domain. The conclusion would be a P2P scientific programming methodology based on experimentations and evaluation on an actual P2P development environment.

5. Software

5.1. XtremWeb

XtremWeb is an open source middleware, generalizing global computing plarforms for a multi-user and multi-parallel programming context. XtremWeb relies on the notion of services to deploy a Desktop Grid based on a 3 tiers architecture. This architecture gathers tree main services: Clients, Coordinators and Workers. Clients submit requests to the coordinator which uses the worker resources to execute the corresponding tasks. Currently tasks concern computation but we are also considering the integration of storage and communication capabilities. Coordinator sub-services provide resource discovery, service construction, service instantiation and data repository for parameters and results. A major concern is fault tolerance. XtremWeb relies on passive replication and message logging to tolerate Clients mobility, Coordinator transient and definitive crashes and Worker volatility. The Client service provides a Java API which unifies the interactions between the applications and the Coordinator. Three client applications are available: the Java API that can be used in any Java applications, a command line (shell like) interface and a web interface allowing users to easily submit requests, consult status of their tasks and retrieve results. A second major issue is the security. The origins of the treats are the applications, the infrastructure, the data (parameters and results) and the participating nodes. Currently XtremWeb provides user authentication, application sandboxing and communication encryption. We have developed deployment tools for harnessing individual PCs, PCs in University or Industry laboratories and PCs in clusters. XtremWeb provides a RPC interface for bag of tasks, parameter sweep, master worker and workflow applications. Associated with MPICH-V, XtremWeb allows the execution of unchanged MPI applications on Desktop Grids.

XtremWeb has been tested extensively harnessing a thousand of Workers and computing a million of tasks. XtremWeb is deployed in several sites: Lille, University of Geneva, University of Tsukuba, University of Paris Sud. In this last site, XtremWeb is the Grid engine of the Paris Sud University Desktop Grid gathering about 500 PCs. Two multi-parametric applications are used in production since the beginning of 2004: Aires belonging to the HEP Auger project and a protein conformation predictor using a molecular dynamic simulator.

The software, papers and presentations are available at www.xtremweb.net.

5.2. MPICHV

Currently, MPICH-V proposes 3 protocols: MPICH-V1, MPICH-V2, MPICH-V/CL. MPICH-V1 implements an original fault tolerant protocol specifically developed for Desktop Grids relying on uncoordinated checkpoint and remote pessimistic message logging. It uses reliable nodes called Channel Memories to store all in transit messages. MPICH-V2 improves the performance of MPICH-V1 (reducing the fault tolerance overhead and increasing the tolerance to node volatility) by implementing a new protocol splitting the message logging into message payload logging and event logging. These two elements are stored separately on the sender node for the message payload and on a reliable event logger for the message events. The third protocol called MPICH-V/CL is derived from the Chandy-Lamport global snapshot algorithm. It implements coordinated checkpoint without message logging. This protocol exhibits less overhead than MPICH-V2 for clusters with low fault frequencies. We are currently designing and developing MPICH-V3 specifically for the Grids. It relies on a new protocol mixing causal message logging and pessimistic remote logging of message events. This is a hierarchical protocol being able to tolerate fault inside Grid sites (inside clusters) and faults of sites (the complete crash of clusters).

In addition to fault tolerant properties, MPICH-V:

1. provides a full runtime environment detecting and re-launching MPI processes in case of faults;
2. works on high performance networks such as Myrinet, Infiniband, etc;
3. allows the migration of a full MPI execution from one cluster to another, even if they are using different high performance networks.

The software, papers and presentations are available at www.lri.fr/~gk/MPICH-V.

5.3. The YML Framework for Parallel Computing on P2P Architectures

The complexity of P2P platforms is important. An end-user cannot manage manually such complexity. We provide a set of tools designed to develop and execute large coarse grain applications on peer-to-peer systems. We developed and did the first experimentations of the YML framework for parallel programming on P2P architectures.

The main part of YML project is a high level language for scientific end-users to develop parallel programs for P2P platforms. This language integrates two different aspects. The first aspect is a component description language. The second aspect is a way to link components: a coordination language called Yvette. This language can express graphs of components. These graphs represent applications. The goal of this split is to manage complex coupled applications on peer-to-peer systems.

We designed a framework to take advantage of YML language. It is composed of two directories and the YML Daemon. The daemon written in JAVA uses information contained in both directories to compile and execute YML applications on top of a peer to peer system. We identify previously the two main roles of the daemon. Each role relies on a specific directory. This strict separation enhances portability of applications and permits optimization during the execution stage in real-time. Currently we provide support for the XtremWeb peer to peer middleware.

To illustrate our approach, we did first experimentations for basic linear algebra routines on an XtermWeb P2P platform with a small number of peers. We did performance evaluations and discussed on the necessity to introduce a new accurate performance model for this new computing paradigm.

YML project was launched at the ASCI CNRS lab in 2001 and is developed now in collaboration with the University of Versailles. YML is under integration into SPIN to propose a GUI ASP.

5.4. The Scientific Programming InterNet (SPIN)

SPIN (Scientific Programming on the InterNet), is a scalable, integrated and interactive set of tools for scientific computations on distributed and heterogeneous environments. These tools create a collaborative environment allowing the access to remote resources.

The goal of SPIN is to provide the following advantages: Platform independence, Flexible parameterization, Incremental capacity growth, Portability and interoperability, and Web integration. The need to develop a tool such as SPIN was recognized by the GRID community of the researchers in scientific domains, such as linear algebra. Since the P2P arrives as a new programming paradigm, the end-users need to have such tools It becomes a real need for the scientific community to make possible the development of scientific applications assembling basic components hiding the architecture and the middleware. Another use of SPIN consists in allowing to build an application from predefined components ("building blocks") existing in the system or developed by the developer. The SPIN users community can collaborate in order to make more and more predefined components available to be shared via the Internet in order to develop new more specialized components or new applications combining existing and new components thanks to the SPIN user interface.

SPIN was launched at ASCI CNRS lab in 1998 and is now developed in collaboration with the University of Versailles, PRiSM lab. SPIN is currently under adaptation to incorporate YML, cf. above. A graduate student is working at Lille in the Grand-Large project to develop one ASP on the Web which integrates SPIN, YML and XtremWeb.

6. New Results

6.1. Large Scale Distributed Systems

- MPICH-V3 preview: A hierarchical fault tolerant MPI for Multi-Cluster Grids [1]
  Among the programming models envisioned for the Grid, explicit message passing following the SPMD paradigm is one of the most studied. However, very few attentions have been paid on fault tolerance (MPICH-GF, FT-MPI and MPICH-V projects are some exceptions). None of the previous works have addressed the question of hierarchy: in Grid, faults may occur within each Grid node or between Grid nodes. MPICH-V3 follows several design constraints: a) providing automatic and transparent fault tolerance (detection and restart), b) allowing the choice of the fault tolerant protocol (coordinated or message logging) within each Grid node (cluster). c) allowing partial checkpointing of a MPI execution, d) avoiding coordinated checkpoint approach at the Grid level. MPICH-V3 uses 3 mechanisms: 1) a global runtime capable of launching MPI processes on Grid nodes (cluster through their batch schedulers), detecting faults and relaunching a set of faulty MPI processes on available nodes, 2) a system allowing the migration of a set of MPI processes between heterogeneous clusters and 3) a MPI execution controller ensuring that a set of restarting MPI processes will reach a state coherent with the full system. In this poster we present the evaluation of three basic mechanisms. First, we show that the latency for small messages is increased compared

to the reference protocol (P4). Second, we consider one of the Grid clusters running a coordinated checkpoint based fault tolerant protocol augmented with a logging mechanism for storing causal dependency information of messages. The evaluation clearly demonstrates that the overhead is not significant for applications with high computation to communication ratio. Finally, we presents the cost of checkpoint/ migration/ restart for different kinds of networks.

- Performance comparison of MPI and OpenMP on shared memory multiprocessors [6]
  When using a shared memory multiprocessor, the programmer faces the selection of the portable programming model which will deliver the best performance. Even if he restricts his choice to the standard programming environments (MPI and OpenMP), he has a choice of a broad range of programming approaches.To help the programmer in his selection, we compare MPI with three OpenMP programming styles (loop level, loop level with large parallel sections, SPMD) using a subset of the NAS benchmark (CG, MG, FT, LU), two dataset sizes (A and B) and two shared memory multiprocessors (IBM SP3 Night Hawk II, SGI Origin 3800). We also present a path from MPI to OpenMP SPMD guiding the programmers starting from an existing MPI code. We present the first SPMD OpenMP version of the NAS benchmark and compare it with other OpenMP versions from independent sources (PBN, SDSC and RWCP). Experimental results demonstrate that OpenMP provides competitive performance compared to MPI for a large set of experimental conditions. However the price of this performance is a strong programming effort on data set adaptation and inter-thread communications. MPI still provides the best performance under some conditions. We present breakdowns of the execution times and measurements of hardware performance counters to explain the performance differences.

- Coordinated Checkpoint versus Message Log for fault tolerant MPI [3]
  MPI is one of the most adopted programming models for Large Clusters and Grid deployments. However, these systems often suffer from network or node failures. This raises the issue of selecting a fault tolerance approach for MPI. Automatic and transparent ones are based on either coordinated checkpointing or message logging associated with uncoordinated checkpoint. They are many protocols, implementations and optimizations for these approaches but few results about their comparison. Coordinated checkpoint has the advantage of a very low overhead on fault free executions. In contrary a message logging protocol systematically adds a significant message transfer penalty. The drawbacks of coordinated checkpoint come from its synchronization cost at checkpoint and restart times. In this paper we implement, evaluate and compare the two kinds of protocols with a special emphasis on their respective performance according to fault frequency. The main conclusion (under our experimental conditions) is that message logging becomes relevant for a large scale cluster from one fault every hour for applications with large dataset.

- MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging [11]
  Execution of MPI applications on clusters and Grid deployments suffering from node and network failures motivates the use of fault tolerant MPI implementations. We present MPICH-V2 (the second protocol of MPICH-V project), an automatic fault tolerant MPI implementation using an innovative protocol that removes the most limiting factor of the pessimistic message logging approach: reliable logging of in transit messages. MPICH-V2 relies on uncoordinated checkpointing, sender based message logging and remote reliable logging of message logical clocks. This paper presents the architecture of MPICH-V2, its theoretical foundation and the performance of the implementation. We compare MPICH-V2 to MPICH-V1 and MPICH-P4 evaluating a) its point-to-point performance, b) the performance for the NAS benchmarks, c) the application performance when many faults occur during the execution. Experimental results demonstrate that MPICH-V2 provides performance close

to MPICH-P4 for applications using large messages while reducing dramatically the number of reliable nodes compared to MPICH-V1.

- Performance comparison of MPI and three OpenMP programming styles on shared memory multiprocessors [12]
When using a shared memory multiprocessor, the programmer faces the selection of the portable programming model which will deliver the best performance. Even if he restricts his choice to the standard programming environments (MPI and OpenMP), he has a choice of a broad range of programming approaches.To help the programmer in his selection, we compare MPI with three OpenMP programming styles (loop level, loop level with large parallel sections, SPMD) using a subset of the NAS benchmark (CG, MG, FT, LU), two dataset sizes (A and B) and two shared memory multiprocessors (IBM SP3 Night Hawk II, SGI Origin 3800). We also present a path from MPI to OpenMP SPMD guiding the programmers starting from an existing MPI code. We present the first SPMD OpenMP version of the NAS benchmark and compare it with other OpenMP versions from independent sources (PBN, SDSC and RWCP). Experimental results demonstrate that OpenMP provides competitive performance compared to MPI for a large set of experimental conditions. However the price of this performance is a strong programming effort on data set adaptation and inter-thread communications. MPI still provides the best performance under some conditions. We present breakdowns of the execution times and measurements of hardware performance counters to explain the performance differences.

- XtremWeb and Condor : sharing resources between Internet connected Condor pools [2]
Grid computing presents two major challenges for deploying large scale applications across wide area networks gathering volunteers PC and clusters/parallel computers as computational resources: security and fault tolerance. This paper presents a lightweight Grid solution for the deployment of multi-parameters applications on a set of clusters protected by firewalls. The system uses a hierarchical design based on Condor for managing each cluster locally and XtremWeb for enabling resource sharing among the clusters. We discuss the security and fault tolerance mechanisms used for this design and demonstrate the usefulness of the approach measuring the performances of a multi-parameters bio-chemistry application deployed on two sites: University of Wisconsin/Madison and Paris South University. This experiment shows that we can efficiently and safely harness the computational power of about 200 PC distributed on two geographic sites.

- Augernome and XtremWeb: Monte Carlos computation on a global computing platform [13]
In this paper, we present XtremWeb, a Global Computing platform used to generate monte carlos showers in Auger, an HEP experiment to study the highest energy cosmic rays at Mallargue-Mendoza, Argentina. XtremWeb main goal, as a Global Computing platform, is to compute distributed applications using idle time of widely interconnected machines. It is especially dedicated to -but not limited to- multi-parameters applications such as monte carlos computations; its security mechanisms ensuring not only hosts integrity but also results certification and its fault tolerant features, encouraged us to test it and, finally, to deploy it as to support our CPU needs to simulate showers. We first introduce Auger computing needs and how Global Computing could help. We then detail XtremWeb architecture and goals. The fourth and last part presents the profits we have gained to choose this platform. We conclude on what could be done next.

- Ontology of Desktop Grids: a Pragmatic View from the XtremWeb Experience [14]
Despite the experience acquired with SETI@home, Distributed.net and the other large scale projects, we still have a limited understanding on key issues related to multi-users/multi-applications desktop Grids.Taxonomy of Dgrids is not so easy to establish. The classification could certainly consider

the system architecture and its main application. It should also consider the deployment/usage parameters. We also face the difficulty that the application domain of DGRID is not well defined yet. Who can you trust if you are a CPU cycle provider, a client or if your role is to control/coordinate the system? Depending on the authentication, tracing and coercion/certification capacities of the system environment, techniques like sandboxing, end to end parameter/code/result encryption and result certification need to be used. The programming model properties are certainly a strong acceptation criterion of DGRIDs. Should we use classical and widely used programming libraries like MPI and RPC, even though they do not provide any means for handling the intrinsic volatility of DGRIDs nodes, or should we consider/design more suited but also more exotic/restricted programming approaches? In this talk, we present our view about these issues acquired during the XtremWeb and MPICH-V projects. We especially detail the third issue, related to programming models.

- High Performance Computing on P2P Platforms: Recent Innovations [17]
  Peer-to-Peer computing systems provide frameworks for running distributed computations and exchanging document/data files between a large set of participants, who can be flexibly and dynamically connected with each other via industry- and university-level networks or via the Internet. Because of the size, autonomy and high volatility of their resources, P2P computing platforms raise major challenging issues (deployment, security, manageability, etc.) for researchers and engineers. But they also propose new fault tolerance and scalability features which raise a lot of expectations and make them potential alternative to classical client-server infrastructures for large-scale systems.

- [16] presents design and implementation of a remote Procedure call (RPC) API for programming applications on Peer-to-Peer environments. The P2P-RPC API is designed to address one of neglected aspect of Peer-to-Peer -the lack of a simple programming interface. In this paper we examine one concrete implementation of the P2P-RPC API derived from OmniRPC (an existing RPC API for the Grid based on Ninf system). This new API is implemented on top of low-level functionalities of the XtremWeb Peer-to-Peer Computing System. The minimal API defined in this paper provides a basic mechanism to make migrate a wide variety of applications using RPC mechanism to the Peer-to-Peer systems. We evaluate P2P-RPC for a numerical application (NAS EP Benchmark) and demonstrate its performance and fault tolerance properties.

6.2. Large Scale Peer to Peer Performance Evaluations

- [18] presents a large scale block-based Gauss-Jordan algorithm to invert very large dense matrices. This version proposes to exploit a peer-to-peer platform. We assume that we access to a scheduler that propose strategies allowing owner computations and data migration anticipation heuristics. We present performance theoretical evaluation results showing that an efficiency of 30% is possible to invert a very large matrix on a platform where peers are heterogeneous and interconnected by a 64 Mbits/s network and with a sufficient number of computers. Nevertheless, the efficiency can drop to only 0.5% for a very slow web-based platform. We discuss that, in this case, the classical evaluation model is not well-adapted to this peer to peer computing paradigm for large scale scientific computing with heterogeneous computers.

6.3. Volatility and Reliability Processing

- Wormhole routing is most common in parallel architectures in which messages are sent in small fragments called flits. It is a lightweight and efficient method of routing messages between parallel processors. Self-stabilization is a technique that guarantees tolerance to transient faults (e.g. memory corruption or communication hazard) for a given protocol. Self-stabilization guarantees that the network recovers to a correct behaviour in finite time, without the need for human intervention. Self-stabilization also guarantees the safety property, meaning that once the network is in a legitimate state, it will remain there until another fault occurs. [7] presents the first self-stabilizing network algorithm in the wormhole routing model, using the unidirectional ring topology. Our solution benefits from wormhole routing by providing high throughput and low latency, and from self-stabilization by ensuring automatic resilience to all possible transient failures.

- [8] highlights the connexions between the formalism of self-stabilizing distributed systems and the formalism of generalized path algebra and asynchronous iterations with delay. We use the later to prove that a local condition on locally executed algorithm (being a strictly idempotent r-operator) ensures self-stabilization of the global system. As a result, a parameterized distributed algorithm applicable to any directed graph topology is proposed. Its function parameter can be instantiated to produce a large class of protocols, which are self-stabilizing at no additional cost.
  The mutual exclusion problem is fundamental in distributed computing, since it permits processors that compete to access a shared resource to be able to synchronize and get exclusive access to the resource (i.e. execute their critical section). It is well known that providing self-stabilization in general uniform networks (e.g. anonymous rings of arbitrary size) can only be probabilistic. However, all existing uniform probabilistic self-stabilizing mutual exclusion algorithms designed to work under an unfair distributed scheduler (that may choose processors to execute their code in an arbitrary manner) suffer from the following common drawback: Once stabilized, there exists no upper bound on time between two successive executions of the critical section at a given processor.

- In [27], we present the first self-stabilizing algorithm that guarantees such a bound (O(n3), where n is the network size) while working using an unfair distributed scheduler. Our algorithm works in an anonymous unidirectional ring of any size and has a polynomial expected stabilization time.

- [28] surveys works that propose self-stabilizing solutions to problems arising in Computer Networks, such as routing and transport layers, or network control protocols. We also review techniques to design self-stabilizing protocols, and mechanisms that reduce the stabilization time when the number of hitting faults is small.

- A routing algorithm is loop-free if, a path being constructed between two processors p and q, any edges cost change induces a modification of the routing tables in such a way that at any time, there always exists a path from p to q. [20] presents a self-stabilizing loop-free routing algorithm that is also route preserving. This last property means that, a tree being constructed, any message sent to the root is received in a bounded amount of time, even in the presence of continuous edge cost changes. Also, and unlike previous approaches, we do not require that a bound on the network diameter is known to the processors that perform the routing algorithm. We guarantee self-stabilization for many metrics (such as minimum distance, shortest path, best transmitter, depth first search metrics, etc.), by reusing previous results on r-operators.

- BGP (Border Gateway Protocol) is the standard inter domain routing protocol in the internet. Inter domain routing permits to route between autonomous systems (such as Universities or companies). BGP is a path vector protocol enriched with policies that permit to alter routing information that is transmitted to other autonomous systems. The problem of inter domain routing stability is to know that, starting from a coherent well known configuration, the protocol converges towards a stable path to every destination. It is well known that this problem is NP-complete or NP-hard depending on the made hypothesis. Several approaches have been defined to propose sufficient conditions for system stability. The problem of inter domain routing self-stabilization is to know that, starting from an arbitrary configuration, the protocol converges towards a stable path to every destination. In [21] and [22], the relationships between the two problems are studied and developped.

8. Other Grants and Activities

8.1. Regional, National and International Actions

- ACI Data Mass Grid eXplorer, 3 years, head: F. Cappello
- Specific Action of CNRS enabling Grid5000, 1 year, F. Cappello
- Global Computing: Augernome XtremWeb, Multi-Disciplinary Project (University of Paris XI), 4 years, sub-projet chair: Franck Cappello
- CNRS-Urbana Champaign Collaboration Program, 1 year, F. Cappello. Visit of Geraud Krawezik at Urbana in November 2003.
- ACI GRID CGP2P: Global Peer to Peer Computing, 3 years, head: F. Cappello
- ACI GRID 2. head: Jean Louis Pazat, sub-topic chair: F. Cappello
- ACI DataGraal. head: Pierre Sens, sub-topic chair: F. Cappello
- Specific Action of CNRS "Analyse Structurelle et Algorithmique des Reseaux Dynamiques" (DY-NAMO), 1 year, head: P. Fraigniaud
- Mobicoop (Agents mobiles cooperatifs pour la recherche dinformations dans des reseaux non fiables) CNRS JemSTIC action, 2 years, head: S. Tixeuil.
- STAR (Stabilisation des reseaux fondes sur la technologie Internet), CNRS JemSTIC action, 2 years, head: S. Tixueil.

8.2. Industrial Contacts

- RNTL Project cASPer : Community based Application Service Provider , 2 years, started in 2001, sub-project chair: F. Cappello
- CEA DAM, between CESTA Bordeaux and LIFL, 8 months, started in August 2002, head: Serge Petiton
- GIE EADS, Thesis founding (CIFRE) for Geraud Krawezik, from November 2001, 3 years. Title: QUID: programming technics and middleware for large computing infrastructures.

10. Bibliography

Major publications by the team in recent years

[1] AURELIEN BOUTEILLER, GERAUD KRAWEZIK, PIERRE LEMARINIER, FRANCK CAPPELLO. *MPICH-V3: A hierarchical fault tolerant MPI for Multi-Cluster Grids.* IEEE/ACM SC 2003, Phoenix USA, November, 2003.

[2] OLEG LODYGENSKY, GILLES FEDAK, FRANCK CAPPELLO, VINCENT NERI, MIRON LIVNY, DOUGLAS THAIN. *XtremWeb and Condor : sharing resources between Internet connected Condor pools.* in « GP2PC 2003 Workshop », IEEE/ACM CCGRID2003, May 12-15, 2003, Tokyo, Japan.

[3] AURELIEN BOUTEILLER, PIERRE LEMARINIER, GERAUD KRAWEZIK, FRANCK CAPPELLO. *Coordinated Checkpoint versus Message Log for fault tolerant MPI.* in « IEEE Cluster 2003 », Hong Kong, December, 2003.

Doctoral dissertations and “Habilitation” theses

[4] GILLES FEDAK. *XtremWeb: an platform for the experimental study of peer-to-peer global computing.* 2003, June, Paris XI University.

[5] THOMAS HERAULT. *Mending of transient failure in self-stabilizing systems.* 2003, June, Paris XI University.

Articles in referred journals and book chapters

[6] GERAUD KRAWEZIK, FRANCK CAPPELLO. *Performance comparison of MPI and OpenMP on shared memory multiprocessors.* in « Journal of Concurrency and Computation: Practice and Experience », Wiley and Sons, 2003.

[7] AJOY K. DATTA, MARIA GRADINARIU, ANTHONY B. KENITZKY, SEBASTIEN TIXEUIL. *Self-stabilizing Wormhole Routing on Ring Networks.* in « Journal of Information Science and Engineering », 2003, volume 19, pages 401-414, Extended abstract in IEEE ICPADS 2002, best paper award.

[8] BERTRAND DUCOURTHIAL, SEBASTIEN TIXEUIL. *Self-stabilization with Path Algebra.* in « Theoretical Computer Science », 2003, volume 293, number 1, pages 219-236, Extended abstract in Sirrocco 2000.

[9] LAURENT ROZAS. *The word problem for 1LC-congruences is NP-hard.* in « Theoretical Computer Science », 2003, volume 306, pages 245-268.

Publications in Conferences and Workshops

[10] DOMINIQUE AMBROISE, BRIGITTE ROZOY, JEAN SAQUET. *Deadlock Detextion in Distributed Systems.* in « the ISCA 18th International Conference on "Computers And Their Applications" », Honolulu, Hawaii, USA, March 26-28, 2003, pages 210-213.

[11] AURELIEN BOUTEILLER, FRANCK CAPPELLO, THOMAS HERAULT, GERAUD KRAWEZIK, PIERRE LEMARINIER, FREDERIC MAGNIETTE. *MPICH-V2: a Fault Tolerant MPI foor Volatile Nodes based on Pessimistic*

Sender Based Message Logging. in « IEEE/ACM SC 2003 », Phoenix USA, November, 2003.

[12] GERAUD KRAWEZIK, FRANCK CAPPELLO. *Performance comparison of MPI and three OpenMP programming styles on shared memory multiprocessors.* in « ACM SPAA 2003 », June 9-7, San Diego, USA.

[13] OLEG LODYHENSKY, GILLES FEDAK, VINCENT NERI, FRANCK CAPPELLO, ALAIN CORDIER. *Auger and XtremWeb: Monte Carlos computation on a global computing platform.* CHEP 2003, in « Conference on Computing in High Energy and Nuclear Physics », March 24-28, 2003, La Jolla, California, USA.

[14] FRANCK CAPPELLO. Invited Presentation, *Ontology of Desktop Grids: a Pragmatic View from the XtremWeb Experience.* DGRID 2003, editors, Phoenix, USA, November, 2003.

[15] FREDERIC MAGNIETTE, LAURENCE PILARD, BRIGITTE ROZOY. *Model-Checking et Produit Synchronise.* in « Conference MSR2003 », Metz.

[16] SAMIR DJILALI. *P2P-RPC: Programming Scientific Applications on Peer-to-Peer Systems with Remote Procedure Call.* Third Workshop on Global and Peer to Peer Computing, editors, Tokyo, Japan, May 12-15, 2003.

[17] FRANCK CAPPELLO. *Invited Presentation, P2P Computing: from expectations to feedback.* Trans-European-Research and Education Networking Association, Mai, 2003, Zagreb, Croatia.

[18] SERGE PETITON, LAMINE AOUAD. *Large Scale Peer to Peer Performance Evaluations, with Gauss-Jordan Method as an example.* in « HeteroPar03 conference », 2003, Septembre, Springer Verlag, LNCSPoland.

[19] SERGE PETITON, OLIVIER DELANNOY. *Block Linear Algebra on the Peer to Peer XtremWeb/YML platform.* in « SIAM conference on Computational Science and Engineering », 2003, February, San Diego.

[20] COLETTE JOHNEN, SEBASTIEN TIXEUIL. *Route Preserving Stabilization.* in « Proceedings of the Sixth Symposium on Self-stabilizing Systems (SSS'03, », pages 183-197, 2003, number 2704, series Lecture Notes in Computer Science, San Francisco, USA, june, Springer Verlag.

[21] SYLVIE DELAET, DUY-SO NGUYEN, SEBASTIEN TIXEUIL. *Stabilite et Auto-stabilisation de BGP.* in « Proceedings of Algotel 2003 », 2003, Banyuls, France, may, INRIA.

[22] SYLVIE DELAET, DUY-SO NGUYEN, SEBASTIEN TIXEUIL. *Stabilite et Auto-stabilisation du Routage Interdomaine dans Internet.* in « Proceedings of RIVF 2003 », pages 139-144, 2003, Hanoi, Vietnam, february, Studia Informatica Universalis.

Internal Reports

[23] TED HERMAN, SEBASTIEN TIXEUIL. *A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks.* Laboratoire de Recherche en Informatique, 2003, number 1370.

[24] SYLVIE DELAET, BERTRAND DUCOURTHIAL, SEBASTIEN TIXEUIL. *Self-stabilization with r-operators in Unreliable Directed Networks.* Laboratoire de Recherche en Informatique, 2003, number 1361.

[25] COLETTE JOHNEN, FRANCK PETIT, SEBASTIEN TIXEUIL. *Auto-stabilisation et Protocoles Reseau.* Laboratoire de Recherche en Informatique, 2003, number 1357.

[26] COLETTE JOHNEN, SEBASTIEN TIXEUIL. *Route Preserving Stabilization.* Laboratoire de Recherche en Informatique, 2003, number 1353.

Miscellaneous

[27] AJOY DATTA, MARIA GRADINARIU, SEBASTIEN TIXEUIL. *Self-stabilizing Mutual Exclusion with Arbitrary Scheduler.* to appear in The Computer Journal, 2003.

[28] COLETTE JOHNEN, FRANCK PETIT, SEBASTIEN TIXEUIL. *Auto-stabilisation et Protocoles Reseaux.* to appear in Technique et Science Informatiques, 2003.

Bibliography in notes

[29] M. J. FISCHER, N. A. LYNCH, M. S. PATERSON. Imposibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382, April 1985.

[30] J. F. SHOCH, J. A. HUPP. *The "Worm" programs: Early Experiences with Distributed Systems.* . Communications of the Association for Computing Machinery 25March, 1982.

[31] M. LITZKOW. *Condor — A hunter of idle workstations.* . In Proceedings of the Eighth Conference on Distributed Computing Systems, San Jose, California.

[32] 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 Practice and Experiencenumber 28(9):929– 961, 199 .

[33] A. BARAK, O. LA'ADAN. *The MOSIX multicomputer operating system for high performance cluster computing.* . Journal of Future Generation Computer Systemsnumber 13(4–5):361–372, 1998.

[34] LUIS F. G. SARMENTA, SATOSHI HIRANO. *Bayanihan: Building and studying web-based volunteer computing systems using Java.* . Future Generation Computer Systemsnumber 15(5-6):675–686, 1999.

[35] A. D. ALEXANDROV, M. IBEL, K. E. SCHAUSER, C. J. SCHEIMAN. *SuperWeb: Research Issues in JavaBased Global Computing.* . Concurrency: Practice and Experiencenumber 9(6):535–553, June, 1997.

[36] T. BRECHT, H. SANDHU, M. SHAN, J. TALBOT. *ParaWeb: Towards World-Wide Supercomputing, in Proc. of the Seventh ACM SIGOPS European Workshop: Systems Support for Worldwide Applications, Sep. 1996.* .

[37] N. CAMIEL, S. LONDON, N. NISAN, O. REGEV. *The POPCORN Project: Distributed Computation over the Internet in Java, 6th International World Wide Web Conference, April 1997* .

[38] H. PEDROSO, L. M. SILVA, J. G. SILVA. *Web-Based Metacomputing with JET, in Proc. of the ACM 1997* .

[39] A. BARATLLO, M. KARAUL, Z. KEDEM, P. WYCHOFF. *Charlotte: Metacomputing on the Web. In In Proceedings of the 9th Conference on Parallel and Distributed Computing Systems, 1996* .

[40] B. O. CHRISTIANSEN, P. CAPPELLO, M. F. IONESCU, M. O NEARY, K. E SCHAUSER, D. WU. *Javelin: Internet-Based Parallel Computing Using Java. Concurrency: Practice and Experience, 9(11):1139–1160, Nov. 1997* .

[41] *http://www.mithral.com/*. http://www.mithral.com/.

[42] *http://boinc.berkeley.edu/*. http://boinc.berkeley.edu/.

[43] G. FEDAK, C. GERMAIN, V. NERI, F. CAPPELLO. *XtremWeb: A Generic Global Computing System. In IEEE Int. Symp. on Cluster Computing and the Grid, 2001* .

[44] ANDREW A. CHIEN, BRAD CALDER, STEPHEN ELBERT, KARAN BHATIA. *Entropia: architecture and performance of an enterprise desktop grid system. J. Parallel Distrib. Comput. 63(5): 597-610 (2003).*

[45] B. UK, M. TAUFER, T. STRICKER, G. SETTANNI, A. CAVALLI. *Implementation and characterization of protein folding on a desktop computational grid - is charmm a suitable candidate for the united devices metaprocessor? Technical Report 385, ETH Zurich, Institute for Comutersystems, October 2002* .

[46] *www.platform.com*. http://www.platform.com.

[47] *www.gridsystems.com* . http://www.gridsystems.com.

[48] *www.datasynapse.com* . http://www.datasynapse.com.

[49] I. FOSTER, A. IAMNITCHI. *On death, taxes, and the convergence of peer-to-peer and grid computing.*. In 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, CA, Feb. 2003.

[50] STEFAN SAROIU, P. KRISHNA GUMMADI, STEVEN D GRIBBLE. *A Measurement Study of Peer-to-Peer File Sharing Systems.* Proceedings of Multimedia Computing and Networking, January 2002 (MMCN'02), San Jose, CA, USA.

[51] N. LEIBOWITZ, M. RIPEANU, A. WIERZBICKI. *Deconstructing the Kazaa Network. in 3rd IEEE Workshop on Internet Applications (WIAPP'03). 2003. Santa Clara, CA* .

[52] S. RATNASAMY, P. FRANCIS, M. HANDLEY, R. KARP, S. SHENKER. *A scalable content-addressable network. In SIGCOMM, Aug. 2001.* .

[53] I. STOICA, R. MORRIS, D. KARGER, F. KAASHOEK, H. BALAKRISHNAN. *Chord: A scalable Peer-To-Peer lookup service for internet applications. In ACM SIGCOMM, Aug. 2001* .

[54] A. ROWSTRON, P. DRUSCHEL. *"Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems", (SIGCOMM 2001 submission)* .

[55] 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, Apr. 2001. .*

[56] J. KUBIATOWICZ, D. BINDEL, Y. CHEN, S. CZERWINSKI, P. EATON, D. GEELS, H. W, FL GUMMADI, S. FI. HEA, W.WEIMER, C. WELLS, B. ZHAO. *Oceanstore: An architecture for global-scale persistent storage. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2000. .*

[57] H. CASANOVA, A. LEGRAND, D. ZAGORODNOV, F. BERMAN. *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In Ninth Heterogeneous Computing Workshop, pages 349-363. IEEE Computer Society Press, 2000. .*

[58] JUNWEI CAO, STEPHEN A. JARVIS, SUBHASH SAINI, GRAHAM R. NUDD. *GridFlow: Workflow Management for Grid Computing. CCGRID 2003.*

[59] A. L. ROSENBERG. *Guidelines for Data-Parallel Cycle-Stealing in Networks of Workstations, I; on maximizing expected output. JPDC, 59:31–53, 99 .*

[60] OTTO SIEVERT, HENRI CASANOVA. *Policies for Swapping MPI Processes. HPDC 2003: 104-113 .*

[61] *Message Passing Interface Forum. MPI: A message passing interface standard. Technical report, University of Tennessee, Knoxville, June 12, 1995. 16 .*

[62] K.M. CHANDY, L. LAMPORT. *Distributed snapshots: Determining global states of distr. systems. ACM Trans. on Comp. Systems, 3(1):63–75, 1985 .*

[63] YI-MIN WANG, W. KENT FUCHS. *: Optimistic Message Logging for Independent Checkpointing in Message-Passing Systems. Symposium on Reliable Distributed Systems 1992: 147-154 .*

[64] L. ALVISI, K. MARZULLO. *"Message Logging: Pessimistic, Optimistic and Causal," Proc. 15th Int'l Conf. on Distributed Com .*

[65] Y. YI, T. PARK, H. Y. YEOM. *A Causal Logging Scheme for Lazy Release Consistent Distributed Shared Memory Systems. In Proc. of the 1998 Int'l Conf. on Parallel and Distributed Systems, Dec. 1998. 1 .*

[66] GEORGE BOSILCA, AURELIEN BOUTEILLER, FRANCK CAPPELLO, SAMIR DJAILALI, GILLES FEDAK, CECILE GERMAIN, THOMAS HERAULT, PIERRE LEMARINIER, OLEG LODYGENSKY, FREDERIC MAGNIETTE, VINCENT NERI, ANTON SELIKHOV. *MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes, in IEEE/ACM SC 2002 .*

[67] *"MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging", in IEEE/ACM SC 2003, Phoenix USA, November 2003. .*

[68] *"Coordinated Checkpoint versus Message Log for fault tolerant MPI", to appear in IEEE Cluster 2003, Hong Kong, December 2003.*

[69] *Simgrid: A Toolkit for the Simulation of Application Scheduling. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid '01), pages 430–437, May 2001. .*

[70] *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, The Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, May 2002 . .*

[71] *Performance evaluation model for scheduling in a global computing system , The International Journal of High Performance Computing Applications, Vol. 14, No. 3, Sage Publications, USA, 2000. .*

[72] *The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations, 1996* . http://www.santafe.edu/projects/swarm/overview/overview.html.

[73] *Globus: A metacomputing infrastructure toolkit, Internat. J. Supercomput. Appl. 11, 2 (1997), 115#128..*

[74] *UNICORE - a Grid computing environment. Concurrency and Computation: Practice and Experience 14(13-15): 1395-1410 (2002) .*

[75] *The physiology of the grid: An open grid services architecture for distributed systems integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002. .*

[76] *The physiology of the grid: An open grid services architecture for distributed systems integration. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002. .*

[77] *Grid Service Specification. Draft 3, Global Grid Forum, July 2002. .*

[78] *www.teragrid.org*. http://www.teragrid.org.

[79] *Use of Multicast in P2P Network thought Integration in MPICH-V2, Master of Science Internship Research Report, Pierre et Marie Curie University, September, 2003..*

[80] *www.scidac.org*. http://www.scidac.org.

[81] *A Science-based Case for Large Scale Simulation, Vol. 1, Office of Science, US Department of Energy, David E. Keyes Report Editor-in-Chief, July 30, 2003.*

[82] *http://www.webservices.org/*. http://www.webservices.org/.

[83] JAN EDLER, ANDREW GOLDBERG, ALLAN GOTTLIEB, SUMEET SOBTI, PETER YIANILOS. *A prototype implementation of archival intermemory. In Proceedings of ACM Digital Libraries. ACM, August 1999..*

[84] LYNCH. *Distributed Algorithms.* Morgan Kaufmann1996.

[85] S. DOLEV. *Self-stabilization, M.I.T. Press 2000.*

[86] V. K. GARG.. *Principles of distributed computing. John Wiley and Sons; ISBN: 0471036005; (May 2002)..*

[87] G. TEL.. *Introduction to distributed algorithms. Cambridge University Press, 2000.*

[88] JOFFROY BEAUQUIER, CHRISTOPHE GENOLINI, SHAY KUTTEN. *Optimal reactive k-stabilization: the case of mutual exclusion. In Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing, pp. 199-208, may 1999 .*

[89] JOFFROY BEAUQUIER, THOMAS HERAULT. *Fault-local stabilization: the shortest path tree. Proceedings of the 21th Symposium of Reliable Distributed Systems, october 2002. .*

[90] CHRISTOPHE GENOLINI, SEBASTIEN TIXEUIL. *A lower bound on k-stabilization in asynchronous systems. Proceedings of the 21th Symposium of Reliable Distributed Systems, october 2002. .*

[91] SHAY KUTTEN, DAVID PELEG. *Fault-local distributed mending. Journal of Algorithms 30(1), pp. 144-165, 1999.*

[92] SHAY KUTTEN, BOAZ PATT-SHAMIR. *Stabilizing time-adaptive protocols. Theoretical Computer Science 220(1), pp. 93-111, 1999.*