Fault Tolerant Implementation of Peer-to-Peer Distributed Iterative Algorithms

The Tung Nguyen, Didier El-Baz
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Université de Toulouse, LAAS, F-31400 Toulouse, France
Email: ttnguyen@laas.fr elbaz@laas.fr

***Abstract*—Fault tolerance issues related to the implementation of distributed iterative algorithms via the P2PDC peer-to-peer distributed computing environment are considered. P2PDC is a decentralized environment dedicated to task parallel applications. It has been designed more particularly for the solution of large scale numerical simulation problems via distributed iterative algorithms. The environment allows frequent and direct communications between peers i.e., machines. P2PDC is based on P2PSAP, a self-adaptive communication protocol. We present new functionalities of P2PDC aimed at making our environment more robust. An adaptive fault tolerance mechanism ensures the robustness of computation to cope with peer faults. We consider also fault tolerance from an algorithmic point of view: we concentrate in particular on distributed asynchronous iterative algorithms that can tolerate some message loss. A series of computational results is presented and analyzed for a numerical simulation problem.**

***Keywords*-distributed computing, peer to peer computing, fault tolerance, task parallel model, numerical simulation.**

I. Introduction

Peer-to-Peer (P2P) applications originally designed for file sharing, e.g., Gnutella [1] or FreeNet [2] are now considered to a larger scope from video streaming and system update to distributed data base and High Performance Computing (HPC) [3]. In this paper, we concentrate on numerical simulation applications. In this context, task parallel model and distributed iterative algorithms i.e., successive approximation methods, give raise to numerous challenges when implemented on P2P networks. We can quote: communication management, scalability, heterogeneity and robustness [4]. Some issues can be addressed by making extensive use of asynchronous iterative schemes, whereby computations are carried out in parallel without order nor synchronization (see [5]). In particular, asynchronous iterative schemes permit one to consider distributed computations whereby peers i.e., machines, go at their own pace according to their intrinsic characteristics and computational load.

In order to obtain good efficiency of P2P HPC applications, new transport protocols have to be designed. In [6], a Peer-To-Peer Self Adaptive communication Protocol (P2PSAP) which is suited to high performance distributed computing has been proposed. P2PSAP is based on the Cactus framework whereby micro protocols can be combined in order to build a desired communication protocol. P2PSAP chooses dynamically the most appropriate communication mode between any peers according to decision taken at application level like scheme of computation, e.g., synchronous or asynchronous scheme and elements of context like network topology at transport level.

In [3], a centralized version of P2PDC an environment for high performance peer-to-peer distributed computing which makes use of the P2PSAP protocol in order to allow direct communication between peers has been presented. P2PDC is devoted to task parallel applications. A first series of computational results obtained for the obstacle problem on the NICTA testbed has also been displayed and analyzed in [3]. A decentralized version of P2PDC with features aimed at making P2PDC more scalable has been presented in [7]. A hybrid topology manager manages peers efficiently and facilitates peers collection for HPC applications. A hierarchical task allocation mechanism accelerates task allocation to peers and avoids connection bottleneck at submitter.

This paper deals with fault tolerance issues in the implementation of distributed iterative algorithms via P2PDC. We present adaptive fault tolerance mechanisms of P2PDC that ensure the robustness of HPC applications and permit one to cope with peer failures. We display and analyze several experimental results obtained for a numerical simulation application.

The paper is structured as follows. Related works are presented in Section II. Some features of the P2PDC environment are recalled in Section III. Section IV is devoted to fault tolerance mechanisms in P2PDC. Section V deals with an example of distributed iterative algorithms for a numerical simulation problem. Experimental results are presented and analyzed in Section VI. Section VII deals with conclusions and future work.

II. Related work

Recently, middleware like BOINC [8] or OurGrid [9] have been developed in order to exploit the CPU cycles of computers connected to the network. Those systems are generally dedicated to applications where tasks are independent and direct communication between machines is not needed. MapReduce [10] is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pair, and a reduce function that merges all intermediate values associated with the same intermediate key. This programming model is not appropriate for distributed iterative algorithms with frequent communication between peers.

P2P-MPI [11] is a framework aimed at developing message-passing programs in large scale distributed networks of computers. P2P-MPI is developed in Java and makes use of Java TCP sockets to implement the MPJ (Message Passing for Java) communication library. P2P-MPI uses a single super-node in order to manage peer registration and discovery that may become a bottleneck. P2P-MPI implements a fault tolerance approach using peer replication that may not be efficient and appropriate to connected problems, since the number of peers involved in the computation will multiply greatly. Furthermore, the coordination protocol ensuring coherence between replicas has great overhead.

III. P2PDC

In this section, we present briefly the decentralized environment P2PDC. Reference is made to [7] and [12] for more details on P2PDC. We recall that P2PDC is an environment for peer-to-peer distributed computing that is devoted to task parallel applications. P2PDC is intended in particular to scientists who want to solve numerical simulation problems via distributed iterative methods (that lead to frequent direct data exchanges between peers). P2PDC relies on the use of the P2PSAP self adaptive communication protocol [6] and a reduced set of communication operations (P2Psend, P2Preceive and P2Pwait) in order to facilitate programming. The programmer cares only about the choice of distributed iterative scheme of computation (synchronous or asynchronous) that he wants to be implemented and does not care about the communication mode between any two peers. The programmer has also the possibility to select an hybrid iterative scheme of computation whereby computations are locally synchronous and asynchronous at the global level. P2PSAP chooses dynamically the most appropriate communication mode between any two peers according to decision taken at application level like scheme of computation and elements of context like network topology at transport level. In the hybrid case, the communication mode between peers in a group of machines that are close and that present the same characteristics is synchronous and the communication mode between peers in different groups is asynchronous.

The decentralized environment P2PDC is based on a hybrid topology manager and a hierarchical task allocation mechanism which make P2PDC scalable. In the sequel, a *task* is relative to a computation submitted to P2PDC and a *subtask* is part of a computation assigned to a given peer.

A. Hybrid topology manager

In the literature, peer-to-peer topologies are designed most of the time for content sharing systems like Chord [13], Pastry [14] or CAN [15].

Computational resources discovery is quite different. Computational resources are specified by peer's characteristics such as CPU, memory, network bandwidth and so on. Hence,

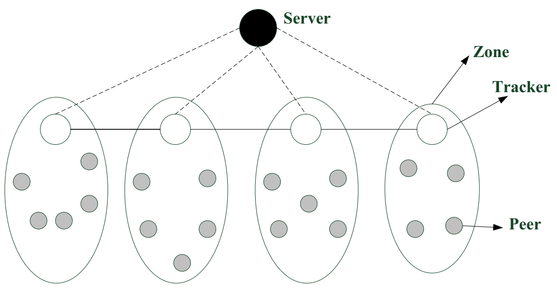

Fig. 1. General topology architecture.

search query in P2P HPC applications may have some specific requirements about peer's characteristics. The requirements may be exact (e.g., CPU speed equals to 3.0 GHz) or in range (e.g., having more than 2Gb of memory). The query will then return the address of N peers that meet the user requirement in order to perform a given computation. Moreover, we note that it is better for peers to be close to each others and to the submitter since the latency is an important factor that influences the efficiency.

The topology manager of P2PDC is based on a simple hybrid architecture which ensures scalability and efficient peer collection for a given HPC application.

1) General topology architecture: Figure 1 illustrates the general topology architecture. It consists of a Server, Trackers and Peers.

- The server manages informations regarding trackers connection and disconnection; it is the contact point of new nodes joining the overlay network for the first time. When a tracker or a peer has no particular contact in order to join the overlay network, it contacts the server in order to receive a list of closest connected trackers; then it connects to trackers in the received list. The server stores also statistic information regarding connection and disconnection time, resources donated or consumed of all nodes of the overlay network.
- A tracker manages information regarding a subset of peers (also called a zone). It collects statistic information regarding connection and disconnection time, resources donated or consumed in its zone and periodically sends these data to the server.
- Peers are donors of computational resources; they are grouped in zones and managed by the tracker of the zone.

Trackers topology is a line (see Figure 2). Each tracker Ti maintains a list of closest trackers Ni in order to ensure that trackers are not isolated. There are $|Ni|/2$ closest trackers having IP address greater than IP address of owner tracker and $|Ni|/2$ closest trackers having IP address smaller than IP address of owner tracker. Moreover, each tracker maintains connection with the closest tracker on its right side and left side.

In a zone, peers publish information regarding processor, memory, hard disk and current usage state to tracker of the zone. Peers have to update periodically their usage state.

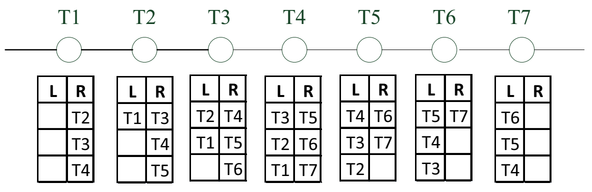

Fig. 2. Trackers topology.

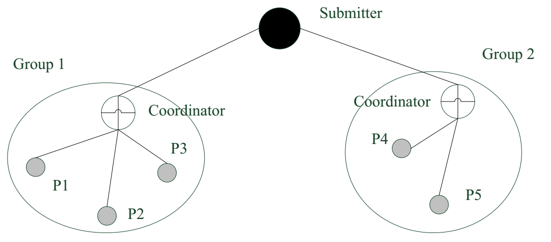

Fig. 3. Allocation graph.

2) IP-based proximity metric: Several proximity metrics can be used in order to calculate the proximity between peers in the network like IP path length, AS path length, geographic distance, and measures related to RTT (see [16]). Each metric presents advantages and drawbacks. IP-based proximity metric has been chosen since it makes use of local information (IP address), it does not consume network resource and it is faster than other metrics.

3) Initial topology: We assume that the system has initially a server and some trackers managed by the system administrator. The core of the system consists of these nodes. When the number of peers increases, the system administrator chooses some reliable volunteers (peers) to become trackers. The choice of trackers relies on on-line time; volunteers with largest on-line time are chosen. Moreover, trackers are chosen so as to ensure that the number of peers in the different zones is well balanced. When the P2PDC environment is downloaded and installed at a node, the IP address of the server and a list of trackers are set and stored in the local memory of the node. The tracker list is updated when the node joins the overlay network.

B. Hierarchical task allocation

When the submitter has collected enough peers, it divides peers into groups taking into account proximity of peers. In each group, a peer is chosen by the submitter to become a coordinator that will manage other peers in the group. The number of peers in a group cannot exceed *Cmax* in order to ensure efficient management of the coordinator. We have chosen *Cmax = 32*. The submitter sends peers list of a group to the coordinator. Then, the coordinator connects to all peers in its group and sends a ”reserve” message to peers. When a peer is reserved for a computation, it sends a message to its tracker in order to inform the tracker that it is not free anymore. Figure 3 illustrates the allocation graph.

The submitter decomposes task into subtasks and sends subtasks to coordinators. Subtasks are then sent by coordinators to peers. Subtasks results are sent in the reverse direction, i.e., peers send subtask results to coordinators, then each coordinator transfers the results to the submitter.

We note that hierarchical task allocation presents many advantages as compared to the case where there is no coordinator. First, hierarchical task allocation is faster since submitter must not connect successively to all peers in order to reserve peers and to send subtasks. Submitter has only to connect to coordinators. Peer reservation and task allocation are carried out in a distributed way by coordinators. Moreover, peers grouping is based on proximity. Hence, communication between coordinator and peers is faster than communication between submitter and peers. Secondly, the transmission of computational results to the submitter via coordinators avoids bottleneck at submitter. Reference is made to [17] for a study on hierarchical framework for grids.

IV. Fault tolerance mechanisms in P2PDC

Peer volatility and in particular peer failure is one of the great challenges in peer-to-peer computing. In peer-to-peer networks, peers may leave the network at unpredictable rate. If a peer assigned to a subtask leaves the network, then, the global task may not terminate or may produce wrong results. Thus, effective fault tolerance mechanisms are vital in order to ensure robustness of the application.

There are two main classes of fault tolerance techniques that can be applied in order to cope with machine failures in distributed and parallel systems: *replication* and *rollback-recovery* (see [18]). In *replication* techniques, each process is replicated on several nodes. A replicated process is called a replica. If some replicas fail, then the other replicas continue to process application. Replication techniques can have two forms: *passive replication* (only a primary replica processes the application, other replicas store backup state of the primary replica) and *active replication* (all replicas process application). *Rollback-recovery* techniques [19] consist in restoring the process of a failed node on another node. Rollback-recovery techniques can be classified into two categories: *checkpoint-based* and *log-based* categories. The *checkpoint-based rollback-recovery* consists in taking a snapshot of the entire system state regularly. Upon a failure, the system is restored to the most recent snapshot. The checkpoint-based rollback-recovery can be classified into three subcategories: *uncoordinated checkpointing*, *coordinated checkpointing* and *communication-induced checkpointing*. In *log-based rollback-recovery*, all communications are logged in a stable storage in addition to process checkpointing, so that upon a failure, only a failed process restores to precedent local checkpoint, performs the same computation as in the initial execution and receives the same messages from stable storage.

During the resolution, peers can have different roles in P2PDC: coordinator or worker. Moreover, computations can be done via different distributed iterative schemes: synchronous, asynchronous or hybrid. Therefore, fault tolerance mechanisms have to adapt themselves to peer roles and computational schemes. In the sequel, we detail the fault tolerance strategies we have selected in each case.

For the kind of application studied here, we consider that replication strategies are not appropriate to workers since they may lead to an important growth of the number of peers without any augmentation of the global computing power. Furthermore, a protocol ensuring coherence between replicas has great overhead when data exchanges between peers are frequent.

Log-based rollback-recovery is not appropriate to distributed iterative algorithms with frequent communications, since communication logging uses a great volume of storage. Thus, we have chosen to carry out a checkpoint-based rollback-recovery mechanism in order to cope with worker failure. This mechanism can adapt itself to different computational schemes. Synchronous iterative schemes need the synchronization of all workers after each iteration, i.e a global state of computation must be reached before computation can continue. Hence, coordinated checkpointing is appropriate to this case. In the asynchronous case, a global state of computation is not needed. Each worker can work at his own pace. Moreover, asynchronous iterative schemes of computation allow message loss. Thus, uncoordinated checkpointing is appropriate to asynchronous iterative schemes. We have implemented a customized checkpointing (programmer defines what data should be placed into checkpoint and how to recover from a checkpoint). The coordinator is only devoted to checkpoint storage of peers in its group since this task may become a bottleneck.

In order to cope with coordinator failure, we have chosen a replication strategy since the number of coordinators is small as compared with the number of workers and coordinators do not compute any subtask. In the following subsections, we present in detail the fault tolerant mechanisms that have been carried out.

1) Checkpoint-based rollback-recovery mechanism for workers: In a group, workers periodically send heartbeat messages to their coordinator in order to inform it that they are still alive. If a coordinator does not receive the heartbeat message from a worker within a time T, then the coordinator considers that this worker has failed.

In order to enable fault tolerant functionality of workers, programmers have to call the *P2P_checkpoint* function in the code when they want workers to take checkpoints. All application data that need to be placed into the checkpoint should be set as parameters of the function. In addition, when a user starts the submitter, he has to add fault tolerance option to command line; otherwise, *P2P_checkpoint* function will have no effect. When fault tolerance option is added, all peers participating to the computation prepare specific data for checkpointing/recovery process; coordinators store a copy of each received subtask so that if a subtask crashes before the first checkpoint, then the coordinator recovers crashed subtask from its initial state. In the sequel, we detail checkpoint-based rollback-recovery processes for the different computational schemes considered. We assume that there are more available free peers than peer failures.

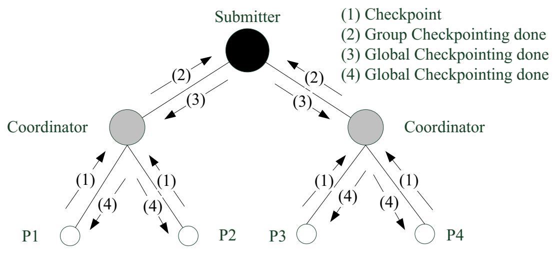

Fig. 4. Coordinated checkpointing process for synchronous scheme.

a) Coordinated checkpointing rollback-recovery for distributed synchronous iterative schemes: Figure 4 displays the different steps of the coordinated checkpointing process in the synchronous case.

- (1) When *P2P_checkpoint* function is called at a worker, the worker creates a checkpoint and sends the checkpoint to its coordinator. We note that in iterative algorithms, all peers execute the same code. Moreover, in the case of synchronous schemes, synchronization between peers is established via blocking operations of communication; thus the *P2P_checkpoint* function is called relatively at the same time on all workers. A worker does not continue to compute immediately after it has sent a checkpoint; it must wait for the consistent global checkpoint of application to be generated.
- (2) Upon reception of a new checkpoint, a coordinator checks if it has received checkpoints from all workers in its group. When all checkpoints have been received, the coordinator notifies the submitter that group checkpointing is done (see Figure 4).
- (3) When the submitter has received notifications from all groups, it generates the consistent global checkpoint of application and notifies all coordinators about the global checkpoint.
- (4) Each coordinator transfers the global checkpoint notification to workers in its group and substitutes old checkpoints in its local memory to new checkpoints. When a worker has received the global checkpoint notification, it replaces the old checkpoint in local memory by the new checkpoint and continues the computation.

If a worker fails, then the workers that exchange data with the failed worker are blocked. Figure 5, displays the case where worker $P4$ fails and a free peer $P5$ is available in the network. The process of rollback and recovery to the last consistent global checkpoint is described below.

- (1) The coordinator of failed worker $P4$ notifies the submitter about peer failure.
- (2) When the submitter has received peer failure notification, it sends *rollback* commands to other coordinators.
- (3) Coordinators transfer *rollback* commands to workers.
- (4) The coordinator of failed peer finds out a free peer in the network, i.e $P5$ and sends failed worker last checkpoint to new peer that is stored in its local memory.

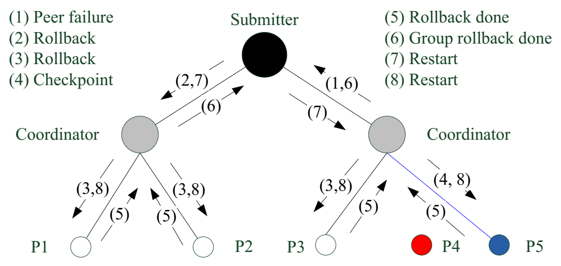

Fig. 5. Recovery process upon a worker failure for synchronous scheme.

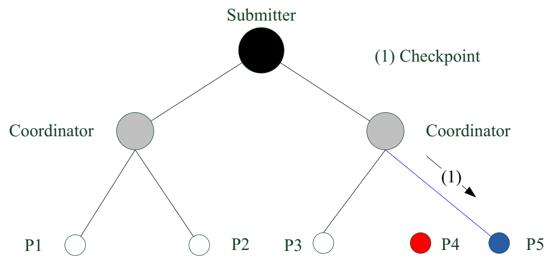

Fig. 7. Recovery process upon a worker failure for asynchronous schemes.

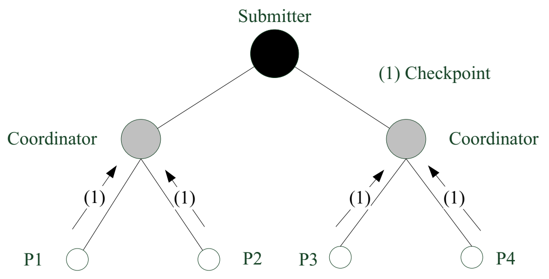

Fig. 6. Uncoordinated checkpointing process for asynchronous schemes.

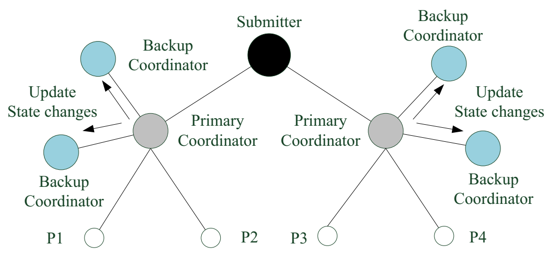

Fig. 8. Replication of coordinators.

- (5) Workers receiving *rollback* command stop computation, load the state of their last checkpoint in local memory and send *rollback done* message to coordinators. Similarly, the new worker $P5$ loads the state from received checkpoint and sends *rollback done* message to coordinator.
- (6) When a coordinator has received *rollback done* messages from all peers in the group, it sends *group rollback done* message to submitter.
- (7) When the submitter has received *group rollback done* messages from all coordinators, it sends *restart* command to all coordinators.
- (8) Coordinators transfer *start computation* messages to workers and workers start computation from the recovered state.

b) Uncoordinated checkpointing rollback-recovery for distributed asynchronous iterative schemes: Figure 6 shows the uncoordinated checkpointing process for asynchronous schemes. The checkpoint process is very simple since no coordination is needed in this case.

- (1) When *P2P_checkpoint* function is called at a worker, the worker creates a checkpoint and sends the checkpoint to its coordinator. Then, the worker continues the computation immediately; moreover the worker does not store the checkpoint in its local memory, contrarily to what is done in the synchronous case. We note that coordinators which receive checkpoints from workers replace old checkpoints by new checkpoints in their local memory.

In the asynchronous case, the recovery process upon a worker failure is very simple, as shown in Figure 7.

- (1) When a worker, e.g., $P4$, fails no worker is blocked, since data exchanges are nonblocking. Thus, other workers continue the computation. The coordinator of the failed worker finds out a free peer in the network, e.g., peer $P5$ and sends the last checkpoint of the worker $P4$ to the peer $P5$. The peer $P5$ loads the state and immediately starts the computation from this state.

2) Replication mechanism for coordinators: A passive replication mechanism is implemented for the coordinators as shown in Figure 8. Each coordinator, also called primary coordinator, has a series of backup coordinators. A primary coordinator communicates with the submitter and workers. Backup coordinators store the state of the primary coordinator. Upon a state change of a primary coordinator, e.g., a new checkpoint or a worker failure, the primary coordinator sends its new state to backup coordinators. Backup coordinators periodically send heartbeat messages to the primary coordinator in order to inform it that they are still alive. When a primary coordinator receives a heartbeat message from a backup coordinator, it sends an acknowledgement message to the backup coordinator. If a primary coordinator does not receive the heartbeat message from a backup coordinator within a time T, then, it considers that this backup coordinator has failed and looks for a free peer in the network to become a new backup coordinator. On the other hand, if backup coordinators do not receive an acknowledgement message from a primary coordinator within a time T, then, they consider that the primary coordinator has failed; backup coordinators elect a backup coordinator to become the new primary coordinator depending on machines load. Finally, the new primary coordinator connects to submitter and workers in the group and starts to manage the group; in addition, the new primary coordinator looks for a free peer in the network to become an additional backup coordinator.

V. Distributed iterative algorithms

This section deals with several examples of distributed iterative algorithms that have been carried out via P2PDC. We concentrate on the solution of the obstacle problem.

A. The obstacle problem

The obstacle problem, belongs to a large class of numerical simulation problems (see [20]). The obstacle problem occurs in many domains like mechanics and finance, e.g., Black-Scholes problem for options pricing.

1) Problem formulation: In the stationary case, the obstacle problem can be formulated as follows.

$$\begin{cases} Find\ u^* \ such\ that \\ A.u^* - f \geq 0, u^* \geq \phi \ everywhere\ in\ domain\ \Omega, \\ (A.u^* - f)(\phi - u^*) = 0 \ everywhere\ in\ \Omega, \\ B.C., \end{cases}$$

where $\phi \in \mathbb{R}^2 (or\ \mathbb{R}^3)$ is an open set, A is an elliptic operator, ϕ a given function and $B.C.$ denotes the boundary conditions on $\partial\Omega$.

In the literature, there are many equivalent formulations of the obstacle problem like complementary problem, variational inequality and constrained optimization problem. We consider here the variational inequality formulation.

$$\begin{cases} Find\ u^* \in K such\ that \\ \forall v \in K, \langle A.u^*, v - u^* \rangle \geq \langle f, v - u^* \rangle, \end{cases}$$

where K is a closed convex set defined by

$$K = \{v | v \geq \phi \ everywhere\ in\ \Omega\},$$

and $\langle ., . \rangle$ denotes the dot product $\langle u, v \rangle = \int uvdx$.

2) Fixed point problem and projected Richardson method: The discretization of the obstacle problem leads to the following fixed point problem whose solution via distributed iterative algorithms presents many interests.

$$\begin{cases} Find\ u^* \in V \ such\ that \\ u^* = F(u^*), \end{cases} \quad (1)$$

where V is an Hilbert space and the mapping $F : v \rightarrow F(v)$ is a fixed point mapping from V into V.

Let α be a positive integer, for all $v \in V$, we consider the following block-decomposition of v and the associated block-decomposition of the mapping F for distributed implementation purpose:

$$\begin{aligned} v &= (v_1, \ldots, v_\alpha), \\ F(v) &= (F_1(v), \ldots, F_\alpha(v)). \end{aligned}$$

We have $V = \Pi_{i=1}^{a} V_i$, where V_i are Hilbert spaces. We denote by $\langle ., . \rangle_i$ the scalar product on V_i and $|.|_i$ the associated norm, $i \in \{1, \ldots, \alpha\}$. For all $u, v \in V$, we denote by $\langle u, v \rangle = \sum_{i=1}^{\alpha} \langle u_i, v_i \rangle_i$, the scalar product on V and $|.|$ the associated norm on V. In the sequel, we shall denote by A a linear continuous operator from V onto V, such that $A.v = (A_1.v, \ldots, A_\alpha.v)$ and which satisfies:

$$\forall i \in \{1, \ldots, \alpha\}, \forall v \in V, \langle A_i.v, v_i \rangle \geq \sum_{j=1}^{\alpha} n_{i,j} |v_i|_i |v_j|_j, \quad (2)$$

where

$$N = (n_{i,j})_{i \leq i, j \leq \alpha} \ is\ an\ M - matrix\ of\ size\ \alpha \times \alpha. \quad (3)$$

The reader is referred to [21] for the definition of $M-matrix$. Similarly, we denote by K_i, a closed convex set such that $K_i \subset V_i, \forall i \in \{1, \ldots, \alpha\}$. We denote by K, the closed convex set such that $K = \Pi_{i=1}^{a} K_i$ and b, a vector of V that can be written as: $b = (b_1, \ldots, b_\alpha)$. For all $v \in V$, let $P_K(v)$ be the projection of v on K such that $P_K(v) = (P_{K_1}(v_1), \ldots, P_{K_\alpha}(v_\alpha))$, where P_{K_i}denotes the mapping that projects elements of V_i onto $K_i, \forall i \in \{1, \ldots, \alpha\}$. For any $\delta \in \mathbb{R}, \delta > 0$, we define the fixed point mapping F_δ as follows (see [20]).

$$\forall v \in V, F_\delta(v) = P_K(v - \delta(A.v - b)). \quad (4)$$

3) Parallel projected Richardson method: We consider the distributed solution of fixed point problem (1) via projected Richardson method combined with several schemes of computation, e.g., a Jacobi like synchronous scheme: $u^{p+1} = F_\delta(u^p), \forall p \in N$ or asynchronous schemes of computation that can be defined mathematically as follows (see [20]).

$$\begin{cases} u_i^{p+1} = F_{i,\delta}(u_1^{\rho_1(p)}, ..., u_\alpha^{\rho_\alpha(p)}) \ if\ i \in s(p), \\ u_i^{p+1} = u_i^p \ if\ i \notin s(p), \end{cases} \quad (5)$$

where

$$\begin{cases} s(p) \subset \{1, \ldots, \alpha\}, s(p) \neq \phi, \forall p \in N, \\ \{p \in N | i \in s(p)\} \ is\ infinite, \forall i \in \{1, \ldots, \alpha\}, \end{cases} \quad (6)$$

and

$$\begin{cases} \rho_j(p) \in N, 0 \leq \rho_j(p) \leq p, \forall j \in \{1, \ldots, \alpha\}, \forall p \in N \\ \lim_{p \to \infty} \rho_j(p) = +\infty, \forall j \in \ \{1, \ldots, \alpha\}. \end{cases} \quad (7)$$

We note that the use of components of the iterate vector that can be delayed in (5) and (7) permits one to model nondeterministic behavior and does not imply inefficacy of the considered distributed scheme of computation. The convergence of asynchronous projected Richardson method has been established in [20]. The reader is referred to [5] and [22] for several studies dealing with convergence issues in asynchronous iterations.

The choice of scheme of computation, i.e., synchronous, asynchronous or any combination of both schemes has important consequences on the efficiency of the distributed iterative algorithm. The interest of asynchronous iterations for various problems including optimization and boundary value problems has been shown in [5], [20], [23], [24], [25].

We note also that asynchronous schemes of computation present some interesting fault tolerance properties since they permit one to cope with message loss that can occur in a peer-to-peer network. In the asynchronous context, message loss is not critic: it does not lead to system deadlock and the information contained in a missing message can be easily replaced via the one contained in a new message.

TABLE I
MACHINE SPECIFICATION AND SEQUENTIAL COMPUTATIONAL TIME

Site	Cluster	Processor	Mem.	Time
Lyon	Sagittaire	AMD 2.4 GHz	2 Gb	32166 s
	Capricorne	AMD 2.0 GHz	2 Gb	33942 s
Sophia	Helios	AMD 2.2 GHz	4 Gb	33178 s
	Sol	AMD 2.6 GHz	4 Gb	29400 s
Toulouse	Pastel	AMD 2.6 GHz	8 Gb	27843 s
Nancy	Grelon	Intel Xeon 1.6 GHz	2 Gb	32476 s
Orsay	Gdx	AMD 2.0/2.4 GHz	2 Gb	34636 s
	Netgdx	AMD 2.0	2 Gb	34711 s

VI. EXPERIMENTAL RESULTS

In this section, we present experimental results obtained with several distributed iterative schemes of computation carried out via P2PDC for a 3D obstacle problem. We concentrate on parallelepiped-based domain decomposition methods. Two cases are considered: a fault-free case and a case with peer failures. In the former case we concentrate essentially on the efficiency of the distributed algorithms carried out with the P2PDC environment; while in the later case, we study the implemented fault tolerance mechanisms.

A. Platform

Experiments have been carried out on the Grid'5000 testbed [26], a French platform, that consists of 2970 processors with a total of 6906 cores distributed over nine sites. Sites of Grid 5000 have several clusters with different characteristics. All sites have at least a Gigabyte Ethernet network for local machines and machines range from 2.5 Gflops up to 10 Gflops.

We have used machines in 8 clusters that belong to 5 sites. Machine characteristics of each cluster and the corresponding sequential computational time are presented in Table I.

The topology server is placed on the Toulouse site. A tracker is launched on each site in order to manage peers of the site. The submitter is a machine of the Sagittaire cluster in Lyon.

B. Computational experiments in a fault-free context

In this subsection, we display and analyze experimental results in a fault-free context. We consider a cubic domain contained in the 3D space, the domain being discretized with 256 x 256 x 256 points.

Table II displays the computational time of distributed iterative schemes, i.e., synchronous, asynchronous and hybrid schemes. Hybrid schemes of computation result from the combination of synchronous and asynchronous schemes: synchronous schemes of computation are carried out locally on peers that are close and present the same characteristics of clock frequency and memory organization (e.g., peers of a given cluster) while the global behavior is asynchronous. We note that when the number of peers is less than 256, computations are carried out on four clusters at four locations, i.e., cluster Pastel at Toulouse, cluster Sagittaire at Lyon, cluster Grelon at Nancy and cluster Gdx at Orsay. For each experiment, an equal number of peers is used at each site.

TABLE II
DISTRIBUTED CASE, COMPUTATIONAL RESULTS

Peers	Method	Time	Speedup	Efficiency
1	-	27843 s	1	1
8	Syn	6152 s	4.53	0.57
8	Asyn	3756 s	7.41	0.93
8	Hybrid	4058 s	6.86	0.86
16	Syn	3222 s	8.64	0.54
16	Asyn	1698 s	16.4	1.03
16	Hybrid	2010 s	13.85	0.87
32	Syn	2049 s	13.59	0.42
32	Asyn	908 s	30.66	0.96
32	Hybrid	1188 s	23.44	0.73
64	Syn	1652 s	16.85	0.26
64	Asyn	457 s	60.93	0.95
64	Hybrid	763 s	36.49	0.57
128	Syn	1338 s	20.81	0.16
128	Asyn	267 s	104.28	0.81
128	Hybrid	510 s	54.59	0.43
256	Syn	1945 s	14.32	0.06
256	Asyn	142 s	196.08	0.77
256	Hybrid	366 s	76.07	0.3

Speedup and efficiency are computed by using the fastest sequential computational time, i.e with a machine in Toulouse.

Experimental results show that synchronous iterative schemes of computation carried out via P2PDC do not scale well up on heterogeneous testbeds. We note also that the combination of asynchronous iterative schemes of computation with P2PDC is very efficient. The lack of synchronization overhead and idle time due to synchronization permits one to obtain very good efficiency. The efficiency of hybrid iterative schemes of computation is situated in between efficiencies of synchronous and asynchronous iterative schemes.

The reader is also referred to [27] for a study related to evolutive problems in Finance.

C. Fault tolerance experiments

In this subsection, we present and analyze experimental results in the case where some peer failures occur. Peer failures are simulated by turning off peers.

1) Coordinator replication overhead: We have run the obstacle problem on 64 workers in the case where the number of backup coordinators varies from 2 to 5. We have found that the overhead of the replication mechanism for coordinators is small. We have also found that coordinator failure has negligible impact on the global solution time. This can be explained by the fact that state changes on primary coordinators are sent to backup coordinators by an independent thread in order to minimize the influence on group management process; moreover, coordinators do not execute any subtask.

2) Worker checkpointing and recovery overhead: We have run the same obstacle problem as in subsection VI-B, i.e., with a domain being discretized with 256 x 256 x 256 points, on 4, 8, 16, 32 and 64 workers. The machines of the cluster Sagittaire at Lyon have been used in the cases with 4, 8, 16 and 32 workers; 32 machines of the cluster Sagittaire at Lyon and 32 machines of the cluster gdx at Orsay have been used in the case with 64 workers. In each case, we have randomly

TABLE III
WORKER CHECKPOINTING AND RECOVERY OVERHEAD

Workers	Checkpoint	Checkpointing time		Recovery time	
	size	Sync	Async	Sync	Async
4	32 Mb	1307 ms	372 ms	1251 ms	1257 ms
8	16 Mb	1349 ms	201 ms	628 ms	654 ms
16	8 Mb	1494 ms	101 ms	320 ms	329 ms
32	4 Mb	1631 ms	51 ms	170 ms	174 ms
64	2 Mb	919 ms	27 ms	105 ms	97 ms

generated worker failures. Table III displays the checkpointing time and recovery time.

The checkpointing time in the synchronous case is greater than the checkpointing time in the asynchronous case. This is due to the fact that in the synchronous case, all worker have to wait for the global checkpoint. Moreover, in the synchronous case, all workers in a group send checkpoints to their coordinator nearly at the same time; while workers send checkpoints to their coordinators at their own pace in the asynchronous case.

In the synchronous case, the checkpointing time generally increases with the number of workers. This is due to the fact that the coordination overhead increases with the number of workers. However, when the number of workers reaches 64 peers, the checkpointing time decreases, since there are two groups (two coordinators) and coordinators receive less checkpoints.

In the asynchronous case, when the number of workers increases, the checkpointing time decreases since the checkpoint size decreases and there is no coordination.

In case of worker failure, the recovery time in the asynchronous context is slightly greater than in the synchronous context (we recall that all workers have to rollback in parallel to the last checkpoint in their local memory in the synchronous case). In the asynchronous case, the coordinator still has to receive checkpoints from other workers while the recovery mechanism is processing; whereas in the synchronous case, only recovery messages are sent. Thus, sending checkpoint of failed workers from the coordinator to a new worker takes more time in the asynchronous case than in the synchronous case. When the number of workers increases from 32 to 64, the recovery time in the synchronous case is greater than in the asynchronous case since machines of two sites, i.e., Lyon and Orsay, have been used and the coordination overhead has consequently increased.

3) Impact of worker failures on computational time: In order to study the impact of worker failures on computational time, we have considered the solution via 64 workers at two locations, i.e., Lyon and Orsay, of the same obstacle problem as in subsection VI-B, i.e., with a domain being discretized with 256 x 256 x 256 points (we recall that in Table II, the case with 64 workers corresponds to machines at four locations). Checkpoints have been taken every 1000 relaxations and some worker failures have been generated randomly. Figure 9 shows the computational time of distributed synchronous

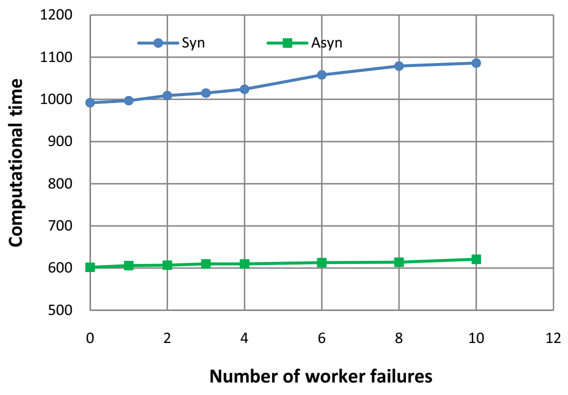

Fig. 9. Computational time for different number of worker failures.

and asynchronous iterative schemes of computation when the number of worker failures varies from 0 to 10.

We note that the computational time increases more with the number of worker failures in the synchronous case than in the asynchronous case. In particular, for ten worker failures, the computational time increases about 10% in the synchronous case and about 4% in the asynchronous case. In the asynchronous case, all peers but one can continue to compute and participate to the application when a peer failure occurs and the fault tolerance mechanism is carried out. In the synchronous case, all peers must carry out a general rollback procedure and must synchronize; meanwhile, they are not available anymore for the application, i.e., computations; moreover, the general rollback mechanism delays the synchronous iterative scheme since computations restart at a previous state, i.e., at a previous iteration number.

The combination of distributed asynchronous iterative schemes of computation with the P2PDC environment seems to be at the same time more efficient in a fault free context and able to cope more efficiently with peer failures. We conclude this section by recalling that asynchronous iterative schemes of computation present also intrinsic fault tolerance properties to deal with messages loss.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a robust version of P2PDC, a decentralized environment for high performance peer-to-peer distributed computing that allows direct communication between peers and that is dedicated to task parallel applications. We have proposed simple and efficient fault tolerance mechanisms ensuring the robustness of HPC applications that adapts to peer roles and computational schemes. We have displayed and analyzed a set of computational results with up to 256 peers on the Grid'5000 testbed for a numerical simulation problem: the obstacle problem. The computational results show that the decentralized and robust version of P2PDC permits one to obtain good efficiency for the solution of numerical simulation problems via distributed iterative methods. In particular, we note that the combination of parallel asynchronous iterative algorithms with P2PDC is very efficient. We have obtained an efficiency close to 80% with 256 peers.

On what concerns robustness aspects to cope with peer faults, we note that the proposed mechanisms have small impact on computational time specially in the asynchronous case. The use of asynchronous distributed iterative algorithms permits also one to cope with message loss.

Presently, we are extending the functionalities of the P2PSAP protocol so as to use Infiniband networks. This will permit us to take into account more testbeds. This will permit us also to reduce solution time. Finally, other applications, e.g., process engineering and logistics will be considered. We plan also to support transparent checkpointing whereby checkpointing/recovering processes are transparent to programmer.

Acknowledgment

Part of this study has been made possible via ANR-07-CIS7-011 grant. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies.

References

[1] "Gnutella protocol development." [Online]. Available: http://rfc.gnutella.sourceforge.net

[2] "The FreeNet network project," http://freenet.sourceforge.net. [Online]. Available: http://freenet.sourceforge.net

[3] T. T. Nguyen, D. El Baz, P. Spiteri, G. Jourjon, and M. Chau, "High performance Peer-to-Peer distributed computing with application to obstacle problem," in *Proceedings of the IEEE Symposium IPDPSW 2010 / HOTP2P*, 2010.

[4] J. Jourjon and D. El Baz, "Some solutions for peer-to-peer global computing," in *Proceedings of the 13th Conference on Parallel, Distributed and network-based Processing*, 2005, pp. 49—58.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. (republished in 1997 by Athena Scientific), 1989.

[6] D. El Baz and T. T. Nguyen, "A self-adaptive communication protocol with application to high performance peer-to-peer distributed computing," in *Proc. of the 18th Conference on Parallel, Distributed and network-based Processing*, 2010.

[7] B. Cornea, J. Bourgeois, T. T. Nguyen, and D. El Baz, "Performance prediction in a decentralized environment for peer-to-peer computing," in *Proceedings of the 25th IEEE Symposium IPDPSW 2011 / HOTP2P 2011*, Anchorage, USA, 2011, pp. 1613—1621.

[8] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *5th IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 4–10.

[9] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg, "OurGrid: an approach to easily assemble grids with equitable resource sharing," in *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, Jun. 2003, pp. 61–68.

[10] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *OSDI*, p. 13, 2004.

[11] S. Genaud and C. Rattanapoka, "A Peer-to-Peer framework for message passing parallel programs," ser. Advances in Parallel Computing, F. Xhafa, Ed. IOS Press, Jun. 2009, vol. 17, pp. 118–147.

[12] T. T. Nguyen, "An environment for peer-to-peer high performance computing," Ph.D. dissertation, University of Toulouse, 2011. [Online]. Available: http://spiderman-2.laas.fr/CIS-CIP/Documents/ThesisNguyen.pdf

[13] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *Journal IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17—32, Feb. 2003.

[14] A. I. T. Rowstron and R. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, 2001, pp. 329—350.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2001, pp. 161—172.

[16] B. Huffaker, M. Fomenkov, D. J. Plummer, D. Moore, and K. Claffy, "Distance metrics in the internet," in *IEEE International Telecommunications Symposium*, 2002, p. 2002.

[17] A. Bendjoudi, N. Melab, and E.-G. Talbi, "An adaptive hierarchical master-worker framework for grids - application to b&b algorithms," *Journal of Parallel and Distributed Computing*, vol. 72, pp. 120—131, february 2012.

[18] M. Treaster, "A survey of Fault-Tolerance and Fault-Recovery techniques in parallel systems," *ACM Computing Research Repository*, vol. 501002, pp. 1—11, 2005.

[19] E. N. M. Elnozahy, L. Alvisi, Y. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, pp. 375—408, Sep. 2002.

[20] P. Spiteri and M. Chau, "Parallel asynchronous richardson method for the solution of obstacle problem," in *Proc. of the 16th Annual International Symposium on High Performance Computing Systems and Applications*, 2002, pp. 133–138.

[21] R. S. Varga, *Matrix Iterative analysis*. Prentice Hall, 1962.

[22] D. El Baz, A. Frommer, and P. Spiteri, "Asynchronous iterations with flexible communication: contracting operators," *Journal of Computational and Appied Mathematics*, vol. 176, pp. 91—103, 2005.

[23] D. El Baz, "M-functions and parallel asynchronous algorithms," *SIAM Journal on Numerical Analysis*, vol. 27, no. 1, pp. 136–140, 1990.

[24] D. Bertsekas and D. El Baz, "Distributed asynchronous relaxation methods for convex network flow problems," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 74–85, 1987.

[25] D. El Baz, "Nonlinear systems of equations and parallel asynchronous iterative algorithms," *Advances in Parallel Computing, vol. 9*, pp. 89–96, 1994.

[26] "Grid5000 platform," http://www.grid5000.fr. [Online]. Available: http://www.grid5000.fr

[27] T. Garcia, M. Chau, T. T. Nguyen, D. El Baz, and P. Spiteri, "Asynchronous peer-to-peer distributed computing for financial applications," in *Proceedings of the IEEE Symposium IPDPSW 2011 / PDSEC*, 2011.