2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies

LBG-SQUARE - Fault-Tolerant, Locality-Aware Co-allocation in P2P Grids

Gérard Dethier[1], Cyril Briquet[1], Pierre Marchot[2], Pierre-Arnoul de Marneffe[1]

[1]Department of Electrical Engineering & Computer Science
[2]Department of Chemical Engineering
University of Liège

{G.Dethier,C.Briquet,Pierre.Marchot,PA.deMarneffe}@ulg.ac.be

Abstract

In this paper, the deployment and execution of Iterative Stencil applications on a P2P Grid middleware are investigated. So-called Iterative Stencil applications are composed of sets of heavily-communicating, long-running Tasks. They thus require co-allocation of multiple reliable resources for extended periods of time.

P2P Grids are totally decentralized and provide on-demand, transparent access to edge resources, e.g. Internet-connected, non-dedicated desktop computers. A P2P Grid has the potential to provide access to a large number of resources at the fraction of the cost of a dedicated cluster. However, edge resources are heterogeneous in performance and intrinsically unreliable: Task execution failures are common due to resource preemption or resource failure. Furthermore, P2P Grid schedulers usually target sets of independent computational Tasks, i.e. so-called Bags of Tasks applications. It is therefore not trivial to deploy and run an Iterative Stencil application on a P2P Grid.

Checkpointing is a common fault-tolerance mechanism in High Performance Distributed Computing, often based on a centralized architecture. Locality-aware co-allocation in P2P Grids has been recently investigated. Checkpointing and locality-aware co-allocation yet have to be integrated in P2P Grids.

We propose to provide co-allocation through an existing middleware-level Bag of Tasks scheduling mechanism. We also introduce a layer of fault-tolerance for the Iterative Stencils that relies on a scalable, application-level, P2P checkpointing mechanism. Finally, LBG-SQUARE is described. This software results from the combination of a specific Iterative Stencil application (a Computational Fluid Dynamics simulation software called LaBoGrid) with a P2P Grid middleware (Lightweight Bartering Grid).

1. Introduction

Peer-to-Peer (P2P) Grid computing, which seeks the convergence of Grid and Peer-to-Peer technologies, is defined as computational resource sharing in Grids organized into P2P networks. A P2P Grid is defined as a transient Virtual Organization that emerges in a bottom-up fashion, out of exchanges of computing time between Peers connected together through a P2P network. Resources providing computing time are managed by Peers. As P2P networks operate at the edge of the Internet, resources are typically edge computers, i.e. unmanaged, not necessarily fast or reliable. The Lightweight Bartering Grid (LBG) [2] is a middleware designed to operate such Grids.

An iterative stencil Application (ISA) can be structured as a set of periodically recomputed interdependent computational Tasks involving heavy and/or frequent communication between subsets of them. They thus require co-allocation of multiple reliable resources for extended periods of time. An example of ISA is distributed Lattice-Boltzmann (LB) simulation. Lattice-Boltzmann Grid (LaBoGrid) [5] is a simulation software adapted to the Grid environment and based on distributed LB methods.

Continuous advances in networks and the increasing availability of numerous edge computers are motivating the computing of ISAs on P2P Grids. Indeed, a P2P Grid has the potential to provide access to a large number of resources at a fraction of the cost of a dedicated cluster.

Resource preemptions and failures are common in a P2P Grid, leading to frequent Task execution failures. Scheduling ISAs in a P2P Grid is not trivial: P2P Grids usually target sets of independent computational Tasks, i.e. so-called Bags of Tasks applications, and resources are usually heterogeneous in performance. Furthermore, ISAs are not easy to deploy because Tasks require dynamic configuration data that are available only at submission time. It is therefore not trivial to deploy and run ISAs on a P2P Grid.

Known load-balancing techniques can be used to provide

978-0-7695-3443-5/08 $25.00 © 2008 IEEE
DOI 10.1109/PDCAT.2008.24

IEEE computer society

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 9, 2009 at 05:26 from IEEE Xplore. Restrictions apply.

locality-aware co-allocation in a P2P Grid. Checkpointing can be used to provide fault tolerance, but requires to be highly scalable and adaptive to a resource environment that can vary at runtime.

The rest of the paper is structured as follows. We show in Section 2 how a P2P Grid middleware can provide locality-aware resource co-allocation for ISAs. We introduce a layer of fault-tolerance for ISAs deployed in a P2P Grid. It is based on a highly scalable, application-level, P2P checkpointing mechanism described in Section 3. The software resulting from the combination of LaBoGrid with LBG, LBG-SQUARE, is described in Section 4. The proposed mechanisms are general and can be adapted to any ISA. Experimental results are detailed in Section 5. A brief conclusion is given in Section 6.

2. Co-allocation for Iterative Stencils

2.1. P2P Grids

A P2P Grid is composed of Peers and resources. Resources are worker computers that run Tasks, and thus provide computing time. As stated in the introduction, resources of a P2P Grid are edge computers that are not necessarily fast or reliable. A Peer manages a set of resources on the behalf of user agents. Peers can share their resources with bartering [4, 2, 3] or market-based methods [3]. Bartering consists of fully distributed, non-monetary exchanges of computing time, and thus does not need a central bank. Peers first use their own resources to compute Tasks submitted by user agents. At peak time, a Peer can consume computing time from other Peers, and supply it back later, at times of low demand levels.

The Lightweight Bartering Grid [2] (LBG) is a recent P2P Grid middleware. Running the LBG middleware enables to make a computer part of the Grid as a Peer or a resource. The Task model in LBG is the Bag of Tasks (BoT), i.e. an application constituted by a set of independent computational Tasks. Task execution is dedicated at the resource level, meaning that at most one Task can be run by a given resource at any time. There is an implicit support for co-allocation in LBG, as a Peer always tries to compute a BoT as fast as possible. When a user agent submits a Bag of Tasks to a Peer, the Peer schedules Tasks to its own resources, preempting the execution of external Tasks (i.e. submitted by other Peers) if necessary, and as defined by its scheduling policy. As long as a Peer has queued Tasks without local resources to run them, it contacts other Peers and asks to submit these Tasks. When a Peer accepts to supply some computing time to compute an external Task, it queues it in a separate queue. If a Peer has queued external Tasks and available resources and no queued local Tasks, it schedules the external Tasks to its available resources.

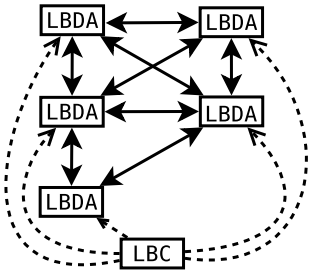

Figure 1. LaBoGrid architecture.

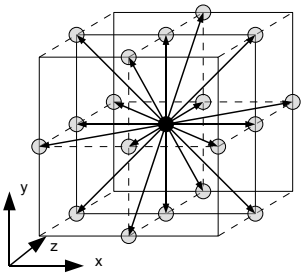

Figure 2. A node of a D3Q19 LB lattice.

The LBG middleware supports Java-programmed Tasks. In practice, any Java application can be easily prepared to be run on the Grid. One Java class is selected as an entry point by implementing a given interface. All Java classes must be packaged into a jar file. Resources run Tasks in a dedicated Java Virtual Machine, separate from the middleware. A security policy is enforced to sandbox Task execution, i.e. to restrict interactions with its environment. Controlled access to a local storage for temporary files is authorized.

2.2. Grid-enabled Lattice-Boltzmann

Computational fluid dynamics (CFD) is a branch of fluid mechanics, based on numerical methods, that deals with problems involving fluid flows. LB simulation methods constitute a family of CFD methods that can deal with complex models and are easily parallelizable. An LB simulation can be structured as an ISA because a stencil application updates every point of a regular grid in function of a weighted subset of nearby points. LB methods use lattices to represent a discretized space, each point of the lattice having some information and being iteratively updated using the current information of the neighboring points. The lattice is partitioned into a set of data blocks, each of which is associated to a Task. Each ISA's Task is connected to a subset of the other ISA's Tasks, also called its neighbors. As these Tasks are tightly interdependent, robust execution is required.

LaBoGrid is a Grid-enabled LB simulation system. It is essentially based on two main software components (see Figure 1): the LB Controller (LBC) and the LB Distributed Agent (LBDA). Each Task of the ISA is actually an LBDA. The LBC generates a computing model from initial parameters. This model is a lattice representing discretized space. Figure 2 shows a node of a 3D lattice and how it is connected to its neighbors (in this case, each node has 18 neighbors). The LBC slices model data into data blocks (pieces of the initial lattice) and distributes them to multiple LBDAs. The sliced model data are the smallest indivisible data units

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 9, 2009 at 05:26 from IEEE Xplore. Restrictions apply.

to process. Performance variability of the resources can be taken into account by balancing data between Tasks.

Efficient load balancing is very important because load imbalance has a strongly negative impact on performance. The slowest LBDA will eventually slow down the whole LaBoGrid. Load rebalancing should thus be done as soon as new resources are used but not too often because of the high cost of updating the state of all LBDAs. A recently proposed algorithm [5] is to balance data between Tasks, according to performance models provided by dynamic benchmarks of available resources.

Deploying LaBoGrid essentially consists of deploying all LBDAs. Once they are deployed and have downloaded initial data from the LBC, LBDAs communicate together at runtime after each iteration of the LB simulation to exchange data. Upon completion, all LBDAs upload results to the LBC, which stores them into a database.

To maximize parallelization of the processing of data blocks, a large number of computers are required. In order to avoid frequent load rebalancing, these computers should all be available from the beginning of the LB simulation until its completion. Co-allocation [6] consists of ensuring simultaneous access to multiple computers for a certain duration.

2.3. Structuring a set of LBDAs as a Bag of (Long-Running) Tasks

Resources can be provided to run the LBDAs by structuring a set of LBDAs as a Bag of Tasks submitted to the Lightweight Bartering Grid middleware (see Figure 3). Running one distributed computing middleware as one Task of another distributed computing middleware is a common pattern. However, it introduces issues related to firewall traversal that are common to all Grid middlewares. The LBG middleware should be able to ask the underlying O.S.-level and/or network-level security infrastructure to temporarily open TCP ports that can be used by Grid applications.

Another issue is that it is very difficult to predict which resource of which Peer will run a given LBDA. It means that the LBC cannot balance the load before LBDAs have been actually deployed on resources. This is why, once deployed, all LBDAs have to first contact the LBC to signal their availability. After having submitted a BoT, the LBC systematically benchmarks the new resources as they become available, including during the execution of the LB simulation. The LBC then performs load balancing and uploads initial data to the deployed LBDAs. Co-allocation is thus augmented with locality-awareness through the combination of benchmarking and subsequent load rebalancing.

Yet another issue is that there is no support for communications between Tasks in the Lightweight Bartering Grid.

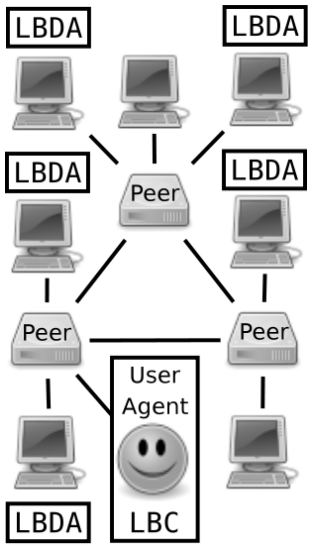

Figure 3. Organization of LBG-SQUARE.

But, as communications are needed between pairs of running Tasks (as opposed to communications between a completed Task and a Task about to be started), LBDAs can directly communicate with one another. To enable this, the LBC communicates to each LBDA the IP address and TCP port of its neighbors along with the initial data.

3. Fault-Tolerance for Iterative Stencils

Task execution failure does not only delay the completion of an ISA (just like a Bag of Tasks), it also suspends the execution of all Tasks (unlike a Bag of Tasks). A common way for ISAs to deal with Task execution failure is the checkpoint/restart mechanism illustrated in Figure 4. A few application-independent fault-tolerance mechanisms for ISAs have been proposed [1, 8, 7] but, to the best of our knowledge, not in P2P Grids.

3.1. Checkpointing and Fault Recovery

The state of every Task is periodically replicated and stored (checkpoint events), possibly multiple times. Checkpointing can be controlled at multiple levels: It could be application-level checkpointing, or transparent (i.e. middleware-level or O.S.-level) checkpointing. Application-level checkpointing must be implemented by Grid application developers. It enables more control and flexibility in the timing, selection and recovery of replicated data [9]. It is thus used in LaBoGrid.

Upon Task execution failure, the execution of the involved Tasks is suspended. This often encompasses all Tasks, not only failed Tasks. The last stored consistent state

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 9, 2009 at 05:26 from IEEE Xplore. Restrictions apply.

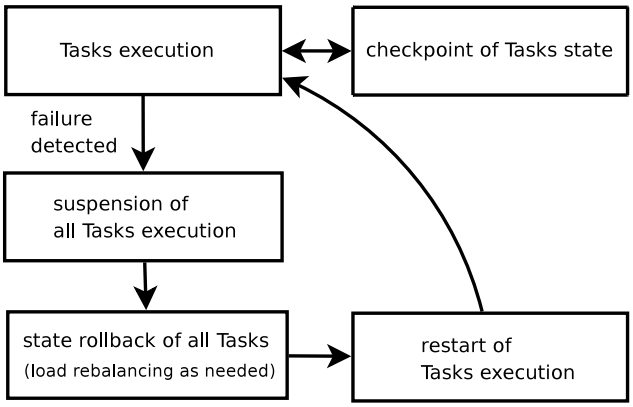

Figure 4. Checkpoint/restart mechanism.

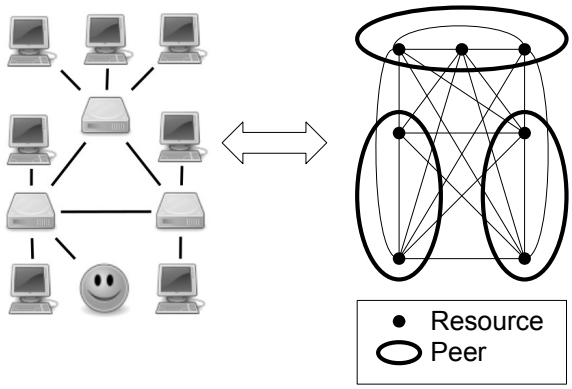

Figure 5. Resource graph of a P2P Grid.

of every Task is reloaded from one of the existing replicas to an available resource. The execution of all Tasks is then restarted.

3.2. Design Parameters

An important design choice is whether to centralize or distribute the storage of Tasks state. Centralized checkpointing is not a good choice as the amount of data that should be stored is very large. More importantly, it is not scalable [8] on large distributed systems like Grids.

The degree of fault-tolerance, i.e. the maximum number of simultaneous Task execution failures that can be tolerated, depends on the number of replicas of each Task. There is a trade-off similar to that of the Error-Correcting Codes: Higher fault-tolerance has a higher cost in terms of data redundancy. Moreover, checkpointing can be disk-based or diskless. Diskless checkpointing [9] avoids the longer access times associated with disk-based data storage. The drawback is the need for larger amounts of available memory.

Given that (1) there is little control off the amount of RAM available on computers at the edge of the Internet and dedicated to the Task (in the order of 500 MBytes of dedicated RAM), (2) the size of LaBoGrid checkpoints can greatly vary (from a few MBytes to more than 100 MBytes) given the amount of available resources and the size of the lattice to be used for a simulation and (3) there would be several replicas to assign to each resource in addition to the current Task state, we currently select disk-based checkpointing.

An approach where disk-based or memory-based checkpointing is chosen given the amount of available RAM can be imagined. However, it is beyond the scope of this paper.

We propose an application-level disk-based checkpointing mechanism, with a P2P checkpointing topology [8], where each resource acts as both a consumer and supplier of replica storage. However, a centralized controller (LBC) has to decide which resource stores the state of its Task on which other resources. In the context of LaBoGrid, the LBC can communicate state replication decisions to all Tasks along with initial data, at no extra cost. Given the robustness/cost trade-off, every LBDA stores a limited number of replicas of its state to other resources. Our proposed checkpointing architecture is thus potentially scalable.

3.3. Management Graphs of LB Simulations

The *model graph* describes the slicing of the initial data of an LB simulation into data blocks, and their interconnections.

The *resource graph* describes the resources obtained from the P2P Grid (see Figure 5) that run the LBDAs. Its nodes are weighted with performance cost (i.e. estimated speed resulting from benchmarking [5]).

The *computation graph* assigns data blocks to resources. It is a partial subgraph of the resource graph. A mapping of the nodes of the model graph to the nodes of the resource graph indicates which resources will be used. Edges of the resource graph are selected according to the connectivity defined by the model graph. These edges are then added to link the selected resources. The mapping of the nodes of the model graph to the nodes of the resource graph is computed in order to minimize processing time. This optimally balances or rebalances the computing times of all Tasks.

The *checkpointing graph* describes the number and location of replicas of the state of every Task. The nodes of the graph are the Tasks. The directed edges indicate to which other Tasks a Task saves its state. It is constructed in two steps: first balancing, at the Peer-level, the number of replicas stored on each resource, then uniformly spreading inbound replicas among the resources of each Peer.

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 9, 2009 at 05:26 from IEEE Xplore. Restrictions apply.

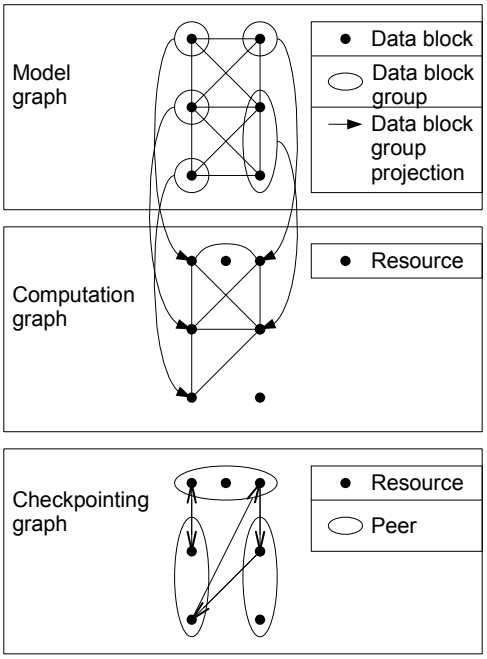

Figure 6. Management graphs.

The model, computation and checkpointing graphs illustrated in Figure 6 are based on the P2P Grid of Figure 3 and resource graph of Figure 5. The nodes of the model graph are data blocks. This graph is non-directed. The nodes of the computation graph are the resources provided by the P2P Grid. This graph is also non-directed. The nodes of the checkpointing graph are the same as the computation graph ones. They are grouped by Peer. This graph is directed.

The resource graph varies over time because of Task execution failures or deployment of new LBDAs. By definition, the computation and checkpointing graphs must also be subsequently recomputed.

Task execution failure can arise from crashes of single resources, or from the simultaneous preemption of multiple resources of a supplier Peer which has an urgent need for its own resources. Therefore, the construction of the checkpointing graph must also ensure that replicas of the state of a Task running on a given resource of a given supplier Peer are preferably stored on resources of other supplier Peers. Even if one Peer preempts all Tasks running on its resources, state replicas will hopefully be available on other Peers' resources.

4. Implementation of LBG-SQUARE

LBG-SQUARE is the software resulting from the combination of the Lightweight Bartering Grid and the LaBoGrid middlewares. It arises from the structuring of a set of LBDAs as a Bag of Tasks. Supplementary coding to support LBG-SQUARE was completed in half a day, which is impressive given the complexity of these two separate middlewares. It essentially involved the writing of a component enabling the LBC to submit Jobs. However, the two middlewares are independent. Load balancing and fault-tolerance are handled by LaBoGrid. Moreover, LBDAs communicate directly with one another, although middleware support could enable the LBDAs to cooperate with the security infrastructure.

4.1. Fault Recovery

In order to provide fault-tolerance upon fault detection, the LBC initiates and coordinates fault recovery operations as explained in Section 3. The LBC temporizes for a short period after detecting a failure of Task execution in order to handle multiple simultaneous Task execution failures. It then submits a new BoT to obtain resources. After a timeout has been exceeded, the computation and checkpointing graphs are recomputed. The LBC then communicates to the LBDAs (including the newly deployed ones) from which other LBDA to download their starting state. The LBC also informs each LBDA of its neighbors in the new computation and checkpointing graphs. When all LBDAs have completed their state rollback, the LBC restarts their execution.

It is important to remark that the described operations are generic to any ISA, and only some implementation details are specific to LaBoGrid. The LBG-SQUARE architecture is completely decentralized under non-faulty conditions. Upon fault detection, the LBC centrally coordinates fault recovery operations. The LBC sends small control messages, which constitute a very low overhead. It is not directly involved in the actual transfers of replicas of Task state; the LBDAs perform these transfers. The LBG-SQUARE architecture is thus fully decentralized for data transfers under faulty conditions. In any case, the LBG-SQUARE architecture is potentially scalable because LBDAs act autonomously and in a P2P fashion.

5. Experimental Results

To illustrate the practical benefits of LBG-SQUARE, distributed checkpointing is compared to centralized checkpointing in terms of execution time of the same Job in a Grid. The execution time is also observed in case of execution failure.

The Lightweight Bartering Grid middleware is deployed on a set of 49 PCs (27 Pentium IV CPU, with 1GB RAM for resources, and 22 Pentium Celeron with 512MB RAM for Peers, user agent and more resources), connected with

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 9, 2009 at 05:26 from IEEE Xplore. Restrictions apply.

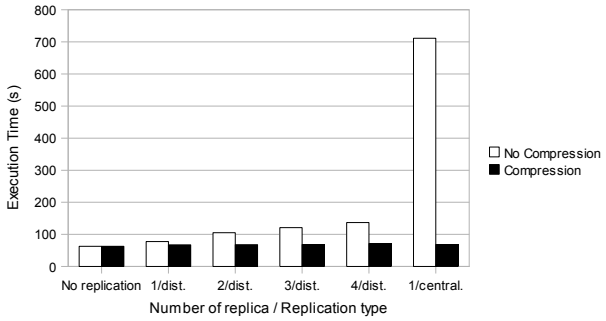

Figure 7. Execution times with distributed and centralized checkpointing.

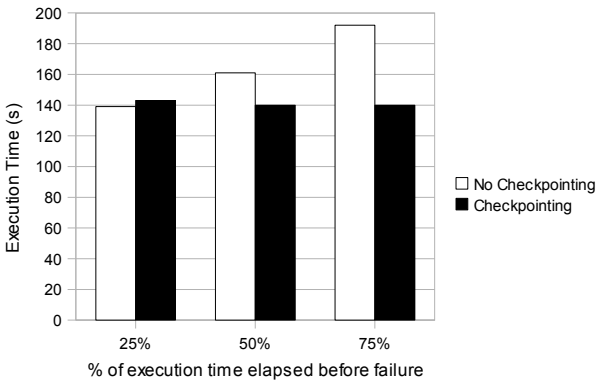

Figure 8. Execution times with execution failure.

a 100 Mbps switched-Ethernet network. A Grid of 8 Peers with 5 resources each is used.

The faulty nature of P2P Grids is simulated by interrupting artificially some tasks running on the resources.

The LaBoGrid Job executed in each experiment is an LB simulation on a lattice of $200 \times 200 \times 200$ nodes. The simulation is run for 100 iterations.

Note that "real-life" simulations use larger lattices and are generally run for 10000 iterations. Thus, they can run for several hours up to a few days. The results presented below show Jobs completed after 3 minutes which is not typical for this kind of application. It is however sufficient to illustrate the behavior of LBG-SQUARE with various replication parameters and failure conditions.

5.1. Distributed and Centralized Checkpointing

The Job is executed using distributed checkpointing and replication degrees 1, 2, 3, and 4. The replication degree bounds the number of Peers that could simultaneously stop supplying resources to the ISA without affecting its completion. The centralized case is a particular case of replication with degree 1 where all Tasks store their replica on the same machine, the LBC's one (which is not part of the Grid). The checkpointing period is fixed at 10 (one checkpoint every 10 iterations). This means there are 10 replications during the Job execution which runs for 100 iterations. Figure 7 gives the execution times of the Job using the given parameters and with compressed and uncompressed state data.

With uncompressed state data (around 20MB per Task state), the distributed checkpointing technique has an overhead in execution time that is much smaller than the overhead with centralized checkpointing. However, comparing cases of uncompressed data, with a replication degree of 4, the execution time is twice as long as the execution time without state replication. Such penalty is probably not acceptable in practice.

When state data are compressed before the replication (around 60KB per Task state, 0.3% of uncompressed data size), the overhead of replication is quite the same in all cases (centralized and distributed checkpointing) and remains very small. The state replication time is dominated by the compression time. The transmission time remains small even with centralized checkpointing. This would not have been the case if there were thousands of Tasks. The gain obtained with compression is very large in our case because of the nature of the compressed data. Compression should be used according to the reachable compression rate.

5.2. Execution Time in Case of Failure

The Job is executed in 2 configurations: with and without checkpointing. An execution failure is caused after a given amount of time in the 2 configurations. Distributed checkpointing is used with replication degree 1 and period 10. The execution failure consists of a very short interruption of 5 resources of a Peer, the resources are immediately available after the interruption and can thus be reused to complete the Job. Figure 8 shows the execution times for execution failures occurring after one quarter, a half and three quarters of the execution time without failure nor checkpointing (see Figure 7).

As expected, without checkpointing, the execution time increases as the execution failure occurs later in the execution. Indeed, more time is wasted (due to lost LB iterations) with a restart from the initial state. With checkpointing, the execution time remains essentially constant. As the Tasks restart from the last saved state, the amount of wasted time remains small all the time. Checkpointing introduces some overhead in execution time with state replication and state rollback but even when the execution failure occurs early, the execution times remain comparable. A long replication

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 9, 2009 at 05:26 from IEEE Xplore. Restrictions apply.

period strongly decreases the execution time but can introduce a larger overhead caused by the reexecution of iterations lost since the last checkpoint.

6. Concluding Remarks

The LBG-SQUARE software has been introduced. It results from the integration of a P2P Grid middleware (Lightweight Bartering Grid [2]) with a specific Iterative Stencil application (Lattice-Boltzmann Grid [5]).

Co-allocation for long-running, heavily-communicating Tasks is provided through a middleware-level Tasks scheduling mechanism intended for independent Tasks. Deploying an Iterative Stencil application (ISA) on a P2P Grid offers opportunities for fault-tolerance, as supplementary resources can be autonomously and dynamically obtained across organizational boundaries. Locality-awareness is achieved through the combination of benchmarking and subsequent load balancing by a controller (LBC).

Application-level agents (LBDAs) are run as independent Tasks on resources provided by the P2P Grid middleware. A central application-level controller initially benchmarks the resources, computes load-balanced computation and checkpointing graphs. It then communicates initial data as well as computation and checkpointing neighbors to the LBDAs. After this initial phase, the computations are fully distributed.

We also introduce a layer of fault-tolerance for ISAs following the checkpoint/restart pattern. It relies on a scalable, application-level, P2P checkpointing mechanism. LBDAs act autonomously and cooperate with one another in a P2P fashion to store replicas of their state. The proposed mechanism is adapted to P2P Grids: The checkpointing graph is constructed to be resilient to multiple simultaneous resource preemptions. The LBG-SQUARE architecture is scalable because it is fully decentralized when the Lattice-Boltzmann simulation is running, including state replication operations. A centralized organization is assumed only during an initial benchmarking and configuration phase, and also upon fault recovery.

State replication can have a strong impact on execution time. However, without a fault-tolerance mechanism, ISAs could not be run on P2P Grids. There is a trade-off between efficiency and robustness: The replication parameters (degree and period) should be selected according to the expected fault levels that the application will face.

Acknowledgments

We want to thank Xavier Dalem for code contributed to the LBG middleware and Valérie Leroy for useful discussions about graph optimization. Some Figures include icons from the Tango library, under Creative Commons Attribution Share-Alike license.

This work was performed in the frame of a research concerted action financed by the Communauté Française de Belgique.

References

[1] C. Banino-Rokkones. *Algorithmic and Scheduling Techniques for Heterogeneous and Distributed Computing*. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, March 2007.

[2] C. Briquet and P.-A. de Marneffe. Description of a Lightweight Bartering Grid Architecture. In *Proc. Cracow Grid Workshop*, Cracow, Poland, 2006.

[3] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. In M. Parashar and C. Lee, editors, *Proc. of the IEEE, Special Issue on Grid Computing*, volume 93, pages 698–714. IEEE Press, NY, USA, March 2005.

[4] W. Cirne, F. Brasileiro, N. Andrade, L. B. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the World, Unite!!! In *J. Grid Computing*. Springer, 2006.

[5] G. Dethier, C. Briquet, P. Marchot, and P.-A. de Marneffe. A Grid-enabled Lattice-Boltzmann-based modelling system. In *Proc. PPAM*, Gdansk, Poland, 2007.

[6] N. Drost, R. V. van Nieuwpoort, and H. E. Bal. Simple Locality-Aware Co-allocation in Peer-to-Peer Supercomputing. In *Proc. GP2P*, Singapore, May 2006.

[7] C. Engelmann and A. Geist. Super-Scalable Algorithms for Computing on 100,000 Processors. In *Proc. ICCS*, Atlanta, GA, USA, May 2005.

[8] C. Engelmann and G. Geist. A Diskless Checkpointing Algorithm for Super-scale Architectures Applied to the Fast Fourier Transform. In *Proc. CLADE'03, HPDC Workshops*, Seattle, WA, USA, June 2003.

[9] J. S. Plank, Y. Kim, and J. J. Dongarra. Fault Tolerant Matrix Operations for Networks of Workstations Using Diskless Checkpointing. In *J. Parallel and Distributed Computing*, volume 43. Elsevier, 1997.

Authorized licensed use limited to: IEEE Xplore. Downloaded on January 9, 2009 at 05:26 from IEEE Xplore. Restrictions apply.