Towards an adaptive middleware for opportunistic environment: a mobile agent approach

Vinicius Pinheiro, Fabio Kon, Alfredo Goldman
Department of Computer Science, University of São Paulo, Brazil
{vinicius,kon,gold}@ime.usp.br

Abstract

The mobile agent paradigm has emerged as a promising alternative to overcome the construction challenges of opportunistic grid environments. This model can be used to implement mechanisms that enable application execution progress even in the presence of failures, such as those presented by the MAG middleware (Mobile Agents for Grids). MAG includes retrying, replication, and checkpointing as fault-tolerance techniques; they operate independently from each other and are not capable of detecting changes on resource availability. In this paper, we describe a MAG extension that is capable of migrating agents when nodes fail, that optimizes application progress by keeping only the most advanced checkpoint, and that migrates slow replicas.

1 Introduction

Opportunistic grids are distributed environments built to leverage the computational power of idle resources geographically spread across different administrative domains. These environments comprise many characteristics such as high level of heterogeneity and large changes on resource availability.

In distributed systems such as opportunistic grids, failures can occur due to several factors, most of them related to resource heterogeneity and distribution. These failures together with the resource usage by its owners modify grid resource availability (i.e., resources can be active, busy, offline, crashed, etc.). The middleware should be able to monitor and detect such changes, rescheduling applications across the available resources and dynamically tuning the fault tolerance mechanisms to better adapt to the execution environment.

In this work, we implemented dynamic fault tolerance mechanisms for grid applications. These mechanisms compose a feedback control system [13, 24], gathering and analyzing information about the execution progress and adjusting its behavior accordingly. To build these mechanisms we rely in the mobile agent paradigm [1]. Mobile agents are programs that can move from one resource to another in an autonomous way, carrying its data and execution state, resuming its execution at the destination. We argue that agents are suitable for constructing opportunistic grids due to intrinsic agent characteristics, such as:

1. *Cooperation*: agents have the ability to interact and cooperate with other agents; this can be explored for the development of complex communication mechanisms among distributed application tasks.

2. *Autonomy*: agents are autonomous entities, meaning that their execution goes on without any or with little intervention from the clients that started them. This is a suitable model for submission and execution of grid applications.

3. *Heterogeneity*: most mobile agent platforms can be executed in heterogeneous environments, an important characteristic for better use of computational resources across multi-organization environments.

4. *Reactivity*: agents can react to external events, such as variations on resources availability.

5. *Mobility*: agents can migrate from one node to another, moving part of the computation being executed, helping to balance the load on grid nodes.

Since 2004, our research group has been using the agent paradigm for developing a grid software infrastructure, leading to the MobiGrid [2] and MAG [19] projects that are based on the InteGrade middleware [14], which follows an opportunistic approach, where workstations idle computing power is used for executing computationally-intensive parallel applications.

This work describes enhancements to the MAG middleware that address the high dynamism of opportunistic grids, managing effectively execution and resource allocation for both sequential and parametric applications.

In the next section, we present some of the related work. In Section 3, we present the MAG architecture and its fault tolerance mechanisms. In Section 4, we describe the implementation of the dynamic replication and unified checkpointing mechanisms. We describe simulation results that assess our proposal in Section 5. Finally, in the last section, we present our conclusions and future work.

2 Related Work

There are several works that are related to this paper, some related to the systems giving support for parametric applications, some related to running long sequential applications on non-dedicated environments, and finally some works are related to the use of mobile agents on grid middleware.

The most well known work was provided by research on extraterrestrial life on the SETI program [23] where more attention were paid on security aspects and on the reliability of the results. More recently, the BOINC project [22] proposed an infrastructure allowing the execution of different programs which can be executed on volunteer computers spread around the world. There exist similar projects both with a fixed algorithm as Mersenne [12], and where different algorithms or challenges can be programmed [10]. However, on these projects the support for long running sequential applications is mostly restricted to local checkpoints (with few exception like [9], or the use of replication to guarantee the progress of the individual applications). Another bag-of-tasks approach is based on OurGrid [7], however the main focus is on dealing with the middleware infrastructure and not on the individual sequential applications.

Several works deals with checkpointing techniques to guarantee the progress of sequential long running applications. One that is directly related to our work is the Grid-WFS framework [15]. In this work the authors studied several approaches to deal with failures on machines. The handling techniques were: retrying, checkpointing, replication, and replication with checkpointing. They concluded that in grid environments with high down-time, as it can happen in opportunistic environments, the replication with checkpointing outperforms the other ones, using as comparison the lower completion time. The Condor project also provides some fault tolerance mechanisms to deal with instable and opportunistic environments: checkpointing and process migration. However, Condor does not perform task replication which would be used to improve application execution progress in the presence of host and network failures.

Several works present the use of mobile agents on grid environments, some using opportunistic contexts (e.g. UWAgents [11]), but most of them presents characteristics more related to the middleware, not the application (e.g.

ARMS [6] and the works published by Loke [17] and Martino and Rana [20]). Some of the mobile agent work were done within our project InteGrade [14]. The first ideas on using mobile agents on an opportunistic grid appeared in [2] where an architecture based on Aglets [16] is first presented, and then evaluated with the use of several replicas in [3]. More recently a work based on the mobile agents framework Jade [25] was also presented [18, 19], where there is application instrumentation, to provide transparent checkpointing and some work on fault tolerance.

To the best of our knowledge this paper is the first one that specifically uses mobile agents combined with replication and checkpointing techniques, within a grid middleware, to provide dynamic fault tolerance mechanisms for sequential and parametric applications on opportunistic environments.

3 The middleware InteGrade/MAG

The InteGrade project involves the development of a grid middleware that leverages the idle computational power of desktop machines. Its architecture follows a hierarchy in which each node can assume different responsibilities. The Cluster Manager is represented by one or more nodes that are responsible for managing that cluster and performing communication with other clusters. A Resource Provider node is the one that exports part of its resources, making them available to grid users. A User Node is one belonging to a grid user who submits grid applications. As we can see in Figure 1, the InteGrade architecture follows a two-tier intra-cluster hierarchy combined with a inter-cluster network.
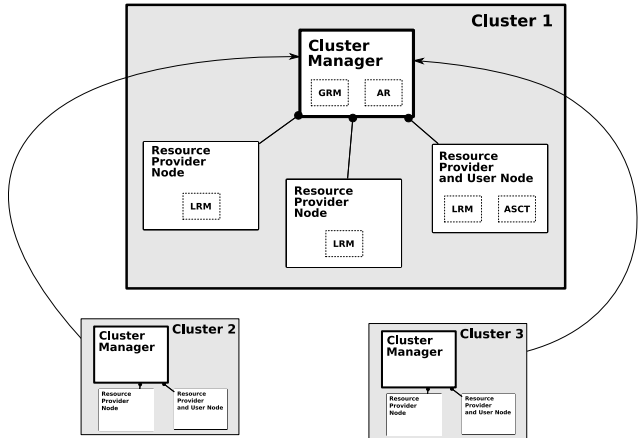
Figure 1. InteGrade architecture

The MAG project [19] introduces the mobile agent technology as a new way of executing applications on InteGrade. Through MAG, the grid user can submit Java applications, not previously supported by the native InteGrade

middleware. This is performed by dynamically loading sequential grid applications into mobile agents. MAG uses JADE (*Java Agent Development Framework*) [25] as the agent platform to provide agent services such as communication and life cycle monitoring. To avoid duplication of efforts, the MAG project was built on top of some InteGrade components: Global Resource Manager (GRM), Local Resource Manager (LRM), Application Repository (AR) and Application Submission and Control Tool (ASCT) (see Figure 1). The GRM is the main grid component and is executed in the Cluster Manager Nodes; it holds information about the registered LRMs and is able to dispatch tasks to them. The LRM is executed in each Resource Provider node; it loads the execution environment and executes tasks submitted to them. The AR provides a cluster repository to store application binaries. Finally, the ASCT provides a user interface for grid application submission, monitoring, and collection of computation results.

In addition, the MAG architecture also adds components that provides mobile agents capabilities and fault-tolerance mechanisms:

1. The *ExecutionManagementAgent (EMA)* stores information about current and past executions, such as current execution state (accepted, running, finished), input arguments, and scheduled machines. This information can be retrieved to restore applications to the point in which they were before the failure.

2. The *AgentHandler* runs on top of the LRMs and works as a proxy to the JADE agent platform, instantiating MAGAgents for each requested execution.

3. The *ClusterReplicationManagerAgent (CRM)* receives requests for execution with replicas from the GRM and creates an ERM agent to handle the request.

4. The *ExecutionReplicationManagerAgent (ERM)* distributes the replicas across the LRMs in the distributed system.

5. The *StableStorage* agent receives the compressed checkpoints, storing them in the file system and retrieving them when prompted. This agent runs in the Cluster Manager node.

6. The *MAGAgent* is the MAG main component; it wraps the application, instantiates it, and catches its exceptions. It also controls application life cycle.

7. The *AgentRecover* is created on demand by the MAG-Agents to recover execution in the presence failures.

3.1 Fault-tolerance in MAG

The MAG fault-tolerance mechanisms can be combined to meet different scenarios of resource availability, resulting in 4 different strategies:

1. *Retrying*: every time the application fails (by throwing an exception), its agent migrates to another node.

2. *Replication*: multiple application replicas are submitted for execution at the same time. When one of the replicas finishes, the others are discarded to avoid waste of resources. In case of failure, retrying is applied.

3. *Checkpointing*: the application periodically saves its execution state in a stable storage. In case of application failure, retrying is applied, but the execution is resumed from the most recent checkpoint.

4. *Checkpointing with Replication*: each replica periodically saves its execution state in a stable storage. Retrying and resuming of execution is applied independently for each replica in the presence of failures.

Currently, the MAG middleware supports only the submission of parametric (bag-of-tasks) and sequential Java applications. This is implemented by extending the `MagApplication` class, wrapping the application code into a mobile agent and submitting it to the agent platform.

In the MAG middleware, the checkpoint mechanism is obtained through code instrumentation provided by the *MAG/Brakes* framework [26].

4 Improving MAG: towards an adaptive middleware

As shown in Section 3.1, the MAG middleware supports multiple fault-tolerance techniques, but these techniques operate solely. Besides, they do not perform any automatic adjustments to adapt themselves to changes in resource availability. If a machine is turned off, for example, all the replicas that were executing on it are lost as MAG only detects failures at the application layer. These replicas are not replaced and the middleware does not make use of their checkpoints.

Events such as network partitioning, crash failures, machine shutdowns, nodes joining the grid, and nodes leaving the grid, define the resource availability of the executing environment. Thus it is desirable that the middleware include fault-tolerance mechanisms to adapt dynamically to these changes.

4.1 Unified Checkpoint

As explained previously, the MAG fault-tolerance mechanisms work independently from each other. This model does not scale well since it makes all replicas perform checkpointing periodically. This increases the communication traffic between resource provider nodes and the stable storage, consuming more grid resources. Another disadvantage of this model is related to resource heterogeneity: in a heterogeneous environment like opportunistic grids some replicas will advance its execution faster than the others. If the most advanced replica crashes in a way that MAG cannot detect, its latest checkpoint will not be used by the slower replicas and part of the execution will be lost.

To resolve this problem, we propose a mechanism named Unified Checkpoint. In this new model, the replicas periodically send information about their execution progress and only the most advanced replica is authorized to perform checkpointing. To enable this feature, the applications must invoke a method which increases a counter. It is in charge of the application programmer to choose the most appropriate places to put these invocations into the code since this is a very application-specific issue. When the replica hits a checkpoint, it sends only the value of the counter and the Stable Storage compares this value to the ones sent by the other replicas. Only the replica with the highest counter value will be requested to save the checkpoint. This model is exemplified in Figure 2.
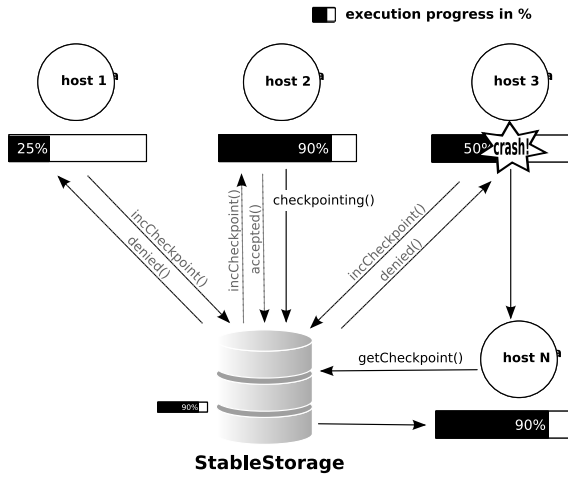

Figure 2. Unified Checkpoint model

In Figure 2, the replica running on host 2 is the most advanced one. When the replica running on host 3 crashes, the MAG recovery mechanism is executed and a new replica is created on host N and the Stable Storage is queried for the checkpoint. The checkpoint stored by the most advanced replica is the only option and so it is sent to the new replica, which resumes its execution from this advanced stage.

4.2 Replica Replacement

Although the checkpointing and the replication of tasks now operate together to form a more integrated fault tolerance system, some events, such as machine crashes, may reduce the number of replicas in execution. Besides, it would be interesting to compare the replica counters to detect replicas that are slow and decide whether they should be moved to another, hopefully faster, computing node

To accomplish that, we propose a feedback control system based on periodical analysis of resource availability. This system is depicted in Figure 3.
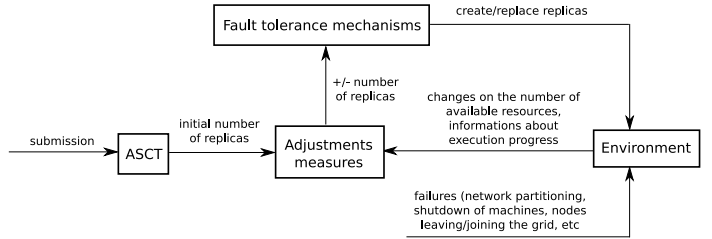
Figure 3. Dynamic replication: a feedback system model

Initially, the grid user submits an application to the grid. The application replicas are created and submitted for execution on the grid nodes. The number of replicas created will be equal to a fixed number defined by the user, but respecting a maximum value of replicas for each application[1]. While running, these replicas are susceptible to failures related to intrinsic characteristics of opportunistic environments such as network partitions, machine shutdown, out-of-memory errors, etc. These failures reduce the number of replicas in execution and also modify the number of available resources. These changes are detected by the system that, after a period without getting responses from the crashed/offline nodes, updates the list of nodes that are still alive (and the new ones that have joined or rejoined the grid recently). Furthermore, during this time, some replicas may become delayed. Based on this information, new replicas are created and late ones are migrated to the new nodes. The Unified Checkpoint will be present throughout this process, so new replicas will resume its execution from the checkpoint of the most advanced replica. This mechanism works even when the most advanced replica crashes, as its last checkpoint remains stored at the Stable Storage so that the new replica can resume from it.

In the current version, the Stable Storage is a single point of failure for our system. This problem could be solve in

---

[1]The maximum number of replicas allowed can be customized by grid administrators

the future by using the Jade agent replication mechanism to implement redundancy in the checkpoint storage.

5   Experiments and simulations

In this section, we present an event-based simulation in various scenarios, demonstrating the potential value of adding dynamic fault-tolerance mechanisms to MAG. Our analysis focuses on task execution times and resources consumption. To run our simulation scenarios, we use the Grid-Sim toolkit [5]. The parameters used in our simulation (mostly borrowed from [4, 21]) follow.

- *Failure rate (λ)* is a random variable representing an arrival rate of failures governed by a Poisson distribution. TBF (time between two adjacent failures) is a random variable governed by an exponential distribution with MTBF representing the mean;

- *Downtime (D)* is the average time following a failure of a task before it is up again, governed by an exponential distribution.

- *Number of replicas (N)* is the number of copies of an application, with each running on different machines.

We simulate a cluster environment with 100 heterogeneous machines connected by a 100Mbps network. The processing power of the resources were generated randomly from 800 to 1600 based on the SPECfp benchmark [8].

We used 3 parameters to model the tasks: number of instructions in MI (millions of instructions), binary size (in bytes) and output file size (in bytes). In our experiments we choose to simulate long running tasks and, to do so, each task submitted has 604800000 MI, the binary size is 320 Kilobytes and the output file size is 15,6 Kilobytes. If we consider the most powerful machine allowed in our experiments, it would take 105 hours to execute the task.

We measured the task execution times to compare the performance of the proposed techniques against the old model. We used 2, 4, 8, and 16 replicas and fixed 60 minutes as the MTBF value to obtain a λ of 24 failures per day. The average downtime was fixed to 30 minutes. These values were used to simulate a very inhospitable environment to distributed processing like student laboratories, where machines are regularly turned off and rebooted. For each number of replicas, we performed 40 simulations, measuring task execution times, computing the arithmetic mean and the 95% confidence interval with a t-Student distribution. We assumed that the application execution time is the execution time of the replica that finishes first. The results are plotted in Figure 4.

First of all, it becomes clear that increasing the number of replicas results in shorter execution times in both strate-
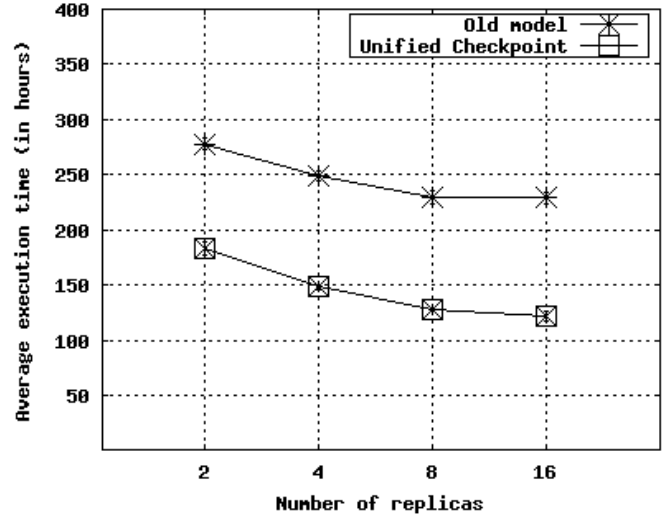

Figure 4. Performance comparison: Unified Checkpoint versus old model

gies. But we can see a considerable gain in the total execution time when using the dynamic strategy presented in this paper. The potential advantage of adopting the Unified Checkpoint mechanism occurs independently of the number of replicas used in our simulation. In all cases, the Unified Checkpoint outperforms the old model obtaining better execution times (at least 34% lower). This difference increases as the number of replicas increases, achieving its maximum performance improvement when 16 replicas were submitted (execution time 47% lower). In the simulated scenarios, which are common in the field of High-Performance Computing, the amount of time saved when using the Unified Checkpoint varied between 95 and 107 hours.

6   Conclusion

Grid middleware hides the complexity related to distribution and heterogeneity and must efficiently address issues such as management and allocation of distributed resources, dynamic task scheduling, fault tolerance, support for high scalability and great heterogeneity of software and hardware components, protection, and security.

The mobile agents paradigm are suitable for dealing with the complexity of building the grid software infrastructure due to its intrinsic characteristics, such as cooperation, autonomy, heterogeneity, reactivity, and mobility. In this work, we present the Unified Checkpoint mechanism, which combines dynamic task replication, replica substitution, and checkpointing to provide fault tolerance for sequential and parametric applications. We use the MAG middleware as the basis for implementing these mechanism.

This middleware benefits from the mobile agent paradigm to encapsulate the applications submitted to the grid.

We demonstrated through our experiments that, in opportunistic environments, it is crucial to support dynamic fault tolerance mechanisms to achieve high performance as well as to make a better use of the available resources in a highly heterogeneous and unstable environment.

Currently, we are investigating other self-optimization and adaptive mechanisms to add to our feedback system. We are currently measuring the benefits of increasing or decreasing the number of replicas dynamically according to three factors: failure rate of the execution environment, number of free resources, and amount of tasks to be scheduled. We are also investigating the impact of changing the checkpointing interval according to the failure rate and the size of the checkpoints to optimize application completion time.

References

[1] Mobile software agents: An overview. *Communications Magazine, IEEE*, 36(7):26–37, July 1998.

[2] R. M. Barbosa and A. Goldman. Framework for mobile agents on computer grid environments. In *In First International Workshop on MATA*, pages 147–157, 2004.

[3] R. M. Barbosa, A. Goldman, and F. Kon. A study of mobile agents liveness properties on mobigrid. In *In 2nd International Workshop on MATA*, 2005.

[4] A. Beguelin, E. Seligman, and P. Stephan. Application level fault tolerance in heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing*, 43:147–155, 1997.

[5] R. Buyya and M. M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CoRR*, cs.DC/0203019, 2002.

[6] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd. Arms: an agent-based resource management system for grid computing. *Scientific Programming (Special Issue on Grid Computing)*, 10(2):135–48, 2002.

[7] W. Cirne, F. Brasileiro, N. Andrade, L. B. Costa, A. Andrade, R. Novaes, and M. Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, September 2006.

[8] S. P. E. Corporation. Specfp 2000 - processor benchmark. http://www.spec.org/cpu2000/results/cfp2000.html. Last accessed on 30 Set, 2009.

[9] CPDN. Do it yourself climate prediction. http://www.climateprediction.net. Last accessed on 27 Feb, 2008.

[10] distributed.net. General-purpose distributed computing project. http://www.distributed.net. Last accessed on 27 Feb, 2008.

[11] M. Fukuda and D. Smith. Uwagents: A mobile agent system optimized for grid computing. In *2006 International Conference on Grid and Applications - CGA'06*, pages 107–113, Las Vegas, NV, Junho 2006.

[12] GIMPS. The great internet mersenne prime search. http://www.mersenne.org. Last accessed on 27 Feb, 2008.

[13] A. Goel, D. Steere, C. Pu, and J. Walpole. Adaptive resource management via modular feedback control. Technical report, Oregon Graduate Institute, Department of Computer Science and Engineering, 1999.

[14] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. C. Bezerra. Integrade: object-oriented grid middleware leveraging the idle computing power of desktop machines. *Concurrency - Practice and Experience*, 16(5):449–459, 2004.

[15] S. Hwang and C. Kesselman. A flexible framework for fault tolerance in the grid. volume 1, pages 251–272, 2003.

[16] IBM. Aglets software development kit. http://www.trl.ibm.com/aglets/. Last accessed on 27 Feb, 2008.

[17] S. W. Loke. Towards data-parallel skeletons for grid computing: An itinerant mobile agent approach. In *Proceedings of the CCGrid'03*, pages 651–652, 2003.

[18] R. F. Lopes. Mag: uma grade computacional baseada em agentes móveis. Master's thesis, Universidade Federal do Maranhão, São Luís, MA, Brasil, January 2006.

[19] R. F. Lopes, F. J. da Silva e Silva, and B. B. Souza. Mag: A mobile agent based computational grid platform. In *Proceedings of CCGrid'05*, LNCS Series, Beijing, November 2005. Springer-Verlag.

[20] B. D. Martino and O. F. Rana. Grid performance and resource management using mobile agents. In *Performance analysis and grid computing*, pages 251–263, Norwell, MA, USA, 2004. Kluwer Academic Publishers.

[21] J. S. Plank and W. R. Elwasif. Experimental assessment of workstation failures and their impact on checkpointing systems. In *28th International Symposium on Fault-Tolerant Computing*, pages 48–57, Munich, June 1998.

[22] SSL. Boinc: The Berkeley open infrastructure for network computing. http://boinc.berkeley.edu/. Last accessed on 27 Feb, 2008.

[23] SSL. Seti@home: Search for extraterrestrial intelligence at home. http://setiathome.ssl.berkeley.edu/. Last accessed on 27 Feb, 2008.

[24] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 145–158, Berkeley, CA, USA, 1999. USENIX Association.

[25] TILAB. Jade - java agent development framework. http://www.jade.tilab.com. Last accessed on 27 Feb, 2008.

[26] E. Truyen, B. Robben, B. Vanhaute, T. Coninx, W. Joosen, and P. Verbaeten. Portable support for transparent thread migration in java. In *In ASA/MA*, LNCS Series, pages 29–43. Springer-Verlag, September 2000.