Adaptive middleware for opportunistic grid: a mobile agent approach

Vinicius Pinheiro, Alfredo Goldman
Dept. of Computer Science, Universidade de São Paulo, Brazil
{vinicius,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 paradigm can be used to implement mechanisms that enable the execution progress of applications even in the presence of failures, such as the mechanisms presented by the middleware MAG (Mobile Agents for Grid Computing Environment). MAG includes retrying, replication and checkpointing as fault-tolerance techniques. However, they operate independently from each other and apart from the changes that may happen in the execution environment. In this paper, we propose a fault-tolerant mechanism based on dynamic task replication and checkpointing in order to meet different scenarios of resource availability.

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 heterogeneity and ~~variation~~ on resource availability.

The central element of a opportunistic grid infrastructure is its middleware. The grid middleware hides the complexity related to distribution ~~and~~ heterogeneity and must efficiently address several 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 requirements.

In distributed systems such as opportunistic grids, failures can occur due to several factors, most of them related to the resources heterogeneity and distribution. These failures ~~among~~ with the usage of the resources by its owners modify the availability status of the resources in the grid (i.e. resources can be active, busy, offline, crashed, etc) and the middleware must monitor~~ing~~ and detect such changes in order to reschedule the applications among the available resources or dynamically tun~~ning~~ the fault tolerance mechanisms to obtain a better adequacy to the actual scenario of availability.

In this work, we implement dynamic fault tolerance mechanism for grid applications and to do so we rely in the mobile agent paradigm. Mobile agents are programs that can move from one resource to another in a autonomous~~ly~~ way carrying its data and execution state to resume its execution at the destination. We argue that these agents exhibits g~~reat~~ adequacy for dealing with the complexity related to the construction of opportunistic grids due to intrinsic 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 grid nodes;
2. *Autonomy*: agents are autonomous entities, meaning that their execution goes on without any or with little intervention by the clients that started them. This is an adequate model for submission and execution of grid applications;
3. *Heterogeneity*: several mobile agent platforms can be executed in heterogeneous environments, an important characteristic for better use of computational resources among multi-institutional environments;
4. *Reactivity*: agents can react to external events, such as variations on resources availability;
5. *Mobility*: mobile agents can migrate from one node to another, moving part of the computation being executed and providing load balancing among grid nodes;
6. *Protection and Security*: several agent platforms offer protection and security mechanisms, such as authentication, cryptography and access control.

Since 2004, our research group has being working on applying the agent paradigm for developing a grid software infrastructure, leading to the MobiGrid and MAG projects [1, 16]. The middleware follows an opportunistic approach, where idle computing power of personal workstations is used for executing computationally-intensive parallel applications.

Two execution models are supported by the developed infrastructure: regular and parametric (or bag-of-tasks) model. The parametric model can be applied to a wide range of grid applications. In this case, multiple copies of a single binary are executed on different grid nodes. Each process receives a sub-set of the application input data and carries out its computation in parallel with the others, without requiring any communication among them.

In this scenario, the likelihood of errors is exacerbated due to several reasons: each process must successfully finish its execution in order to generate a successful application execution; grid applications usually perform complex computations, requiring large execution times; opportunistic grid environments are very dynamic since the user can request at any time exclusive use of resources (workstations).

1.1 Contributions and Paper Organization

This work describes improvements to the MAG grid middleware for efficiently addressing the high dynamic of the opportunistic environments, providing an effective management for long sequential and parametric applications and the allocation of resources necessary for its successful execution.

On the next section we present some of the related work. Then on Section 3 we present the MAG architecture and its fault tolerance mechanisms. On Section 4 we describe the implementation of the dynamic replication and the unified checkpointing mechanisms. We provide some experimental results to evaluate our proposal on Section **??**. Finally, on the last Section, we present our conclusions and next goals.

2. Related Work

There are several works that are related to this paper, some related to the systems giving support for BoT like 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 [19] where more attention were paid on security aspects and on the reliability of the results. More recently, the BOINC project [18] 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 [10], and where different algorithms or challenges can be programmed [7]. However, on these projects the support for long running sequential applications is mostly restricted to local checkpoints (with few exception like [6], or the use of replication to guarantee the progress of the individual applications). Another bag-of-tasks approach is based on OurGrid [5], 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 [12]. 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.

Several works present the use of mobile agents on grid environments, some using opportunistic contexts [9], but most of them presents characteristics more related to the middleware, not the application [3, 4, 14, 17]. Some of the mobile agent work were done within our project InteGrade [11]. The first ideas on using mobile agents on an opportunistic grid appeared in [1] where an architecture based on Aglets [13] is first presented, and then evaluated with the use of several replicas in [2]. More recently a work based on the mobile agents framework Jade [20] was also presented [15, 16], 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 techniques of replication and checkpointing, within a grid middleware, to provide dynamic fault tolerance mechanisms for sequential and parametric applications on opportunistic environments.

3. The middleware MAG

The InteGrade project evolves the development of a grid middleware that leverages the idle computational power of desktop machines. This project is maintained by the Institute of Mathematics and Statistics of the University of São Paulo along with others institutions. InteGrade is based on CORBA [22], an industry standard for distributed object systems. InteGrade naming service uses CORBA IDL (Interface Definition Language) being accessible from a large

variety of programming languages and operating systems. The InteGrade architecture follows an 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, InteGrade architecture follows a two-tier intra-cluster hierarchy, and a peer-to-peer based inter-cluster network.

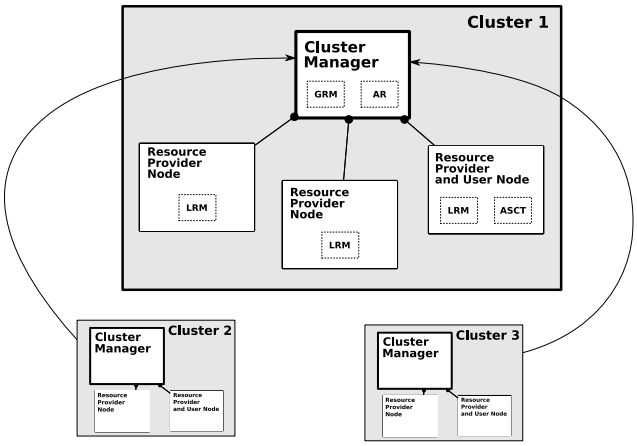

Figure 1. InteGrade architecture

The MAG project was developed by the Dept. of Computer Science of the Federal University of Maranhão and introduces the mobile agent technology as new way of executing applications on InteGrade. Through MAG, the grid user can submit Java applications, not supported by the native InteGrade middleware. This is performed by dynamically loading sequential grid applications into mobile agents. Figure 2 depicts the architecture layers of the MAG middleware. The JADE [20] (*Java Agent Development Framework*) layer represents the agent platform used by MAG to provide agent services such as communication and life cycle monitoring. Jade provides a private queue of messages to each agent allowing them to exchange messages specifying their topics and their receivers. JADE is portable, since it was implemented in Java, and complies with the FIPA [8] specification (Foundation for Intelligent Physical Agents).

In order to avoid duplication of efforts, the MAG project was build on top of some InteGrade components:

1. *Local Resource Manager (LRM)*: This component is executed in each Resource Provider node, loading the execution environment and wrapping the application processes delegated to it. LRM also contains a sub-component called LUPA (Local User Pattern Analyzer) that collects informations about memory and CPU utilization. The LRM sends status information about applications in execution. It also sends heartbeat messages periodically.

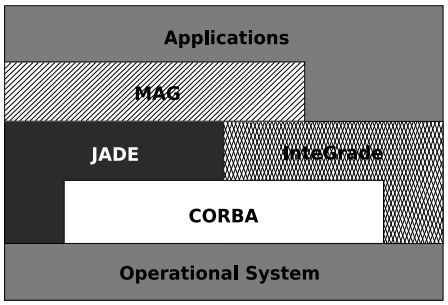

Figure 2. InteGrade/MAG middleware layers

2. *Global Resource Manager (GRM)*: This is the main component in the grid and is executed in the Cluster Manager Node. The GRM holds informations about the registered LRMs and is able to send tasks to them.

3. *Application Repository (AR)*: This component provides a centralized repository to store application binaries that will be submitted for execution.

4. *Application Submission and Control Tool (ASCT)*: This component is executed in the User Node and provides a interface in which the user can submit applications to the grid. The user can also monitoring the execution status and view the results of the execution.

Besides those, MAG architecture adds some others components that provides mobile agents capabilities and fault-tolerance mechanisms:

1. *ExecutionManagementAgent (EMA)*: This component stores informations about current and past executions, as the current execution state (accepted, running, finished), input arguments and scheduled machines. This informations could be retrieved to restore applications to the point that they were before the failure.

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

3. *ClusterReplicationManagerAgent (CRM)*: When the GRM receives a execution with replicas request it delegates to the CRM. This component processes informations for each replica an create an ERM agent to handle the request.

4. *ExecutionReplicationManagerAgent (ERM)*: This component contacts the LRMs of the target machines in order to execute the replicas, one in each machine.

5. *StableStorage*: The stable storage receives the compressed checkpoints in order to store them in the file system, and retrieve it when receives a query request. This agent runs in the Cluster Manager node.

6. *MAGAgent*: This is the main component of the MAG middleware. The MAGAgent wraps the application, instantiate it, and catch the application exceptions that may be raised. It also controls the applications life cycle.

7. *AgentRecover*: This component is created on demand to perform the recovery of the execution in the presence of incidental failures.

3.1 Fault-tolerance in MAG

In this section we present the fault-tolerance mechanisms available on MAG. This mechanisms can be combined to meet different scenarios of resource availability, resulting in 4 different strategies:

1. *Retrying*: Every time the application fails it is automatically submitted again.

2. *Replication*: Various replicas of the application are submitted for execution at the same time. When one of the replicas finishes, the others are discarded to avoid overconsumption 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, the retrying is applied, but the execution is resumed from the most recently checkpoint state.

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

   Currently, the MAG middleware supports the submission of Java applications. In order to execute them, its necessary to extends the `MagApplication` class. This is necessary, so the application code can be wrapped into a mobile agent and submitted to the agent platform. Next, we describe what happens in case of application submission with replicas on MAG (figure 3):

   The user submits the application through the ASCT interface along with informations about its execution (1): input arguments, number of replicas, input and output

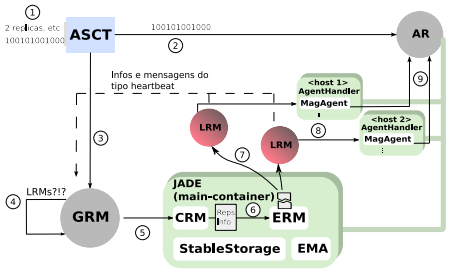

Figure 3. Application submission on MAG

files, etc. The application binary is stored in the AR component (2) and the execution request is sent to the GRM (3). After the submission, the GRM checks it there are sufficient resources (e.g. number of replicas must not exceed the number of LRM nodes) (4). If so, the GRM delegates the execution to the CRM (5). The CRM generates an unique id for each replica and creates an ERM agent to manage the request (6). The ERM proceeds with execution by requesting the LRM's of the target machines (7). Then, each LRM delegates the execution to the AgentHandler which creates a *MAGAgent* for each request (8). The *MAGAgent* is responsible for downloading the application binary from the AR component(9), instantiate the application, and notify the AgentHandler when the execution has finished.

All the informations about execution (e.g. execution time, number of replicas, machines used, etc) are placed onto a relational database by the EMA, and can be queried later. The *GRM* select the *LRM*'s to execute the tasks performing a round-robin strategy.

In a opportunistic environment many problems can arise during the application execution process, e.g. machines being turned off, out of memory errors, heisenbugs, etc. This is even more tricky when executing long running applications, since they are exposed to these problems during a long period of time. When such a failure occurs in the executing environment, the `MAGAgent` class overrides an *uncaughtException* method to handle it. This method instantiate a local Agent Recover that gather information from the GRM in order to get a reference to an AgentHandler of an available node. Finally, the Agent Recover requests the remote AgentHandler to restore the execution. It is worthwhile to say that this retrying mechanism is provided also to every replica, in case of execution with replicas.

In the MAG middleware the checkpoint mechanism is obtained through code instrumentation. This is provided by the *MAG/Brakes* framework. This frame-

work is a modified version of the Brakes framework [21], developed by the DistriNet research group, from the Katholieke Universiteit Leuven, Belgium. The MAG/Brakes framework performs the capture of the state execution of JAVA threads allowing them to resume its executing in another location. The MAG/Brakes also is the core of a powerful migration mechanism, since the execution can be interrupted at any time and be resumed without loss of computational work. This feature saves efforts from the application developers in the sense that there is no need to modified the application code to explicit where the checkpoint must be performed. Currently, the MAG/Brakes only performs code instrumentation of JAVA applications compiled with previous versions of the Java language (1.4 or lower).

When an instrumented application is executing in MAG, it periodically invokes the *setCompressedCheckpoint* method of the `MAGAgent` class. Through this method the agent interacts with the *StableStorage* component. This component is in charge of getting all the compressed execution state and stored it in a local file. When the application execution needs to be restored (e.g. a failure happens), the *getCompressedCheckpoint* method is invoked to restore the application execution state. After that, the application execution is resumed.

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 don't perform any automatic adjustments to adapt themselves regarding changes that may happen in the execution environment. If a machine is turned off, for example, all the replicas that were executing in it are lost because MAG only detect application failures. These replicas are not replaced by the middleware.

Events such as network partitioning, crash failures, shutdown of machines, nodes joining the grid, nodes leaving the grid, etc, define the resource availability of the executing environment and this may change according to the frequency of those events. Hence, we propose that MAG should automatically and dynamically tune its fault-tolerance mechanisms to fit to those changes whenever they happen.

One can argue that a system administrator may observe the changes that happen with the resources at the execution environment and alter the behavior of these mechanisms by changing its execution parameters accordingly, through a interface developed for that aim. But this is a very complex task since it require very specialized knowledge and full time observation from the administrators. Furthermore, in the current version of MAG, reload these parameters would require the shutdown and the restart of all middleware components running in the grid.

4.1 Unified checkpointing

4.2 Dynamic replication

5. Experiments and simulations

In this section, we present event based simulations in diverse scenarios demonstrating the potential value of adding dynamic fault-tolerance mechanisms into MAG. Our analysis will be focused on the execution times of the tasks and the amount of resources used to execute them.

5.1 Methodology

6 Conclusion

References

[1] 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.

[2] 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.

[3] 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.

[4] J. Cao, D. J. Kerbyson, and G. R. Nudd. High performance service discovery in large-scale multi-agent and mobile-agent systems. In *International Journal of Software Engineering and Knowledge Engineering, Special Issue on Multi-Agent Systems and Mobile Agents.*, number 11 in 5, pages 621–641. World Scientific Publishing, 2001.

[5] 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.

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

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

[8] F. for Intellinget Physical Agents. Fipa - foundation for intellinget physical agents. `http://www.fipa.org`. Last accessed on 27 Feb, 2008.

[9] M. Fukuda, Y. Tanaka, N. Suzuki, L. F. Bic, and S. Kobayashi. A mobile-agent-based pc grid. *Autonomic Computing Workshop, 2003*, pages 142–150, 2003.

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

[11] 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.

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

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

[14] 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.

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

[16] 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.

[17] 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.

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

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

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

[21] 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.

[22] S. Vinoski. Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14:46–55, 1997.