A Study of Mobile Agents Liveness Properties on MobiGrid[⋆]

Rodrigo M. Barbosa, Alfredo Goldman, and Fabio Kon

Department of Computer Science
Institute of Mathematics and Statistics
University of São Paulo
{rodbar, gold, kon}@ime.usp.br

Abstract. MobiGrid is a framework for mobile agents support within a grid environment project. The mobile agents may be used to encapsulate long-processing applications (*tasks*). In MobiGrid, to provide a fast method of releasing the local machine and to prevent the *tasks* from dying suddenly, it is possible to have two or more independent copies of these *tasks* running independently; this concept is defined as *liveness*. In this paper, our goal is to simulate a network of personal workstations and the *tasks* that will be executed on them. In this way, we can study the effects of different *liveness degrees* on the completion time of the *tasks*.

1 Motivation

The InteGrade project [1] is building a middleware infrastructure to enable the use of idle processing time of machines already owned by public or private institutions. One of InteGrade's goals is to use this idle time to solve many kinds of parallelizable, computationally intensive problems, including strongly coupled applications.

The MobiGrid project [2] consists in the implementation of a mobile agents infrastructure for InteGrade. The main idea of our framework is to allow an efficient utilization of computational resources by providing a programming environment for long running applications, which we call *tasks*. Among the applications that could be executed using mobile agents, there are loosely coupled parallel applications like SETI@home [3], besides sequential applications that demand long processing time.

The mobile agents migration capability meets two major InteGrade goals: (1) the system must be transparent in terms of performance to the machine user, i.e. the local machine user must have the highest priority compared to InteGrade applications and (2) the idle resources must be used in the best possible way. One of the main goals of InteGrade is to make the system transparent to the local user in a way that the machine is totally available for him when he needs it. There are a few options for dealing with a mobile *task* when the resource is

[⋆] Project supported by The State of São Paulo Research Foundation. Process number: 02/11165-6

no longer available: migration, termination or suspension. The migration option itself could cause the local user to face a performance loss, since migration may be a costly process: for instance, a *task* carrying a big amount of data. When there is only one copy of each *task*, by using the termination option, we would need remote *checkpointing* - this could be a costly solution and was not implemented. Finally, as we work in a highly dynamic environment where the resources might be busy for a long time, we do not allow suspension.

Therefore, MobiGrid has implemented another strategy which is an alternative to migration: the possibility of having multiple copies of each *task*. We think that this solution is simpler and lighter, besides working as a recovery strategy. In the case where there is the necessity to free the machine resources for the local user, if this machine is executing a *task*, the local MobiGrid environment could be quickly killed, since we have two or more copies of this *task* running independently on different machines. We call this property *liveness* and it is a very important concept in our work. When the MobiGrid infrastructure notices the death of a *task*, one of the remaining copies is cloned and then migrated to another machine. In this process, the InteGrade architecture provides information about the network and the other machines, allowing the mobile agent to choose a machine with available computational resources. To do that, usage patterns of other machine resources can also be used.

The objective of this paper is to present a comparison of two strategies that have been used in MobiGrid to free the machine for the local user: migration and cloning with *liveness*. To do that, we have carried out experiments comparing cloning and migration. Based on these experiments, we propose a mathematical model, whose objective is to simulate a network of homogeneous machines and *tasks* executing on them. The main goal of this simulation is to study the *liveness* and its impact on the system. Our model is a simplification of the reality, but we believe that it reflects the expected behavior in practice.

This paper is organized as follows: Section 1 explained the motivation of our work; Section 2 introduces the MobiGrid infrastructure and explains the *liveness* concept; Section 3 describes the mathematical model that we used to perform the simulations; Section 4 presents simulation results; Section 5 concludes this paper.

2 MobiGrid Overview

MobiGrid [2] is a Java framework for mobile agents support on grids based on Aglets [4]. Using Aglets for grid computing is not a new idea. Aversa et al. [5] have already used this environment in a case study, implementing a dynamically load-balanced distributed version of an algorithm for the (0 - 1) *knapsack problem* - a combinatorial optimization problem.

Figure 1 depicts an overview of our framework. Each of the *clients* hosts a *manager*, which communicates with InteGrade. Note that, in this figure, the *liveness degree* - i.e. the number of *twins* of a submitted *task* - is 2. Client Node 1's *manager* manages Task 1 and its clone. Client Node 2's *manager* manages Task

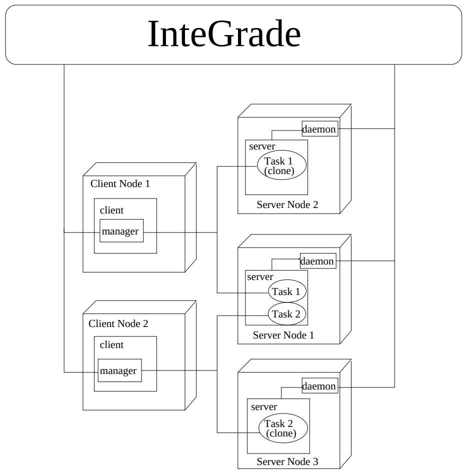

Fig. 1. General architecture of the framework

2 and its clone. Observe also that the *daemons* communicate with InteGrade to inform when a local machine is idle, turning on its server.

3 Mathematical Model

As said before, our goal is to simulate a network of homogeneous machines and *tasks* executing on them in order to study the effects of different *liveness degrees*. Thus, we need a model for this simulation. A model is not supposed to be a perfect map of reality, otherwise, its implementation could be unfeasible. Based on that, we need some simplification assumptions for it.

Our experiments led us to two key simplification assumptions in our model: (1) the cost of a cloning a *task* is zero and (2) the migration of a clone *task* does not interfere with the execution time of the cloned *task* executing on the same *server*. The assumption (1) is derived from the fact that the cloning cost is insignificant compared to the migration cost (about 200 times faster in our experiments). Thus, if the number of cloning operations is not high in the local *server*, the cloning cost is negligible. This fact can be easily explained since the migration process depends on slow I/O operations and the cloning process depends on fast operations on RAM. Moreover, as the cloning in our model is always associated with a migration, in a normal situation, the cloning time can be added to the migration time. Assumption (2) is endorsed by the fact that the migration of a cloned *task* hosted in a *server* does not slows downs significantly (up to 10%) the execution of the cloned *task*. So, if the number of migrating *tasks* in the local *server* is not very high, we may not consider the interference.

As stated above, these assumptions are true if the number of clone operations and migrations occurring in the local *server* is not very high. Thus, to limit these numbers, we imposed two restrictions: (1) the model allows only one *task* per *machine*; this is reasonable for our model as the tasks that will be used on our system are mainly long processing tasks, which work independently based on their own data; if more that one task is executed on a machine, these tasks will slow down; (2) if the *task* of a *machine* was cloned and this clone is migrating from that *machine*, during this migration time, this cloned *task* cannot be cloned again. At this point, a important question may arise: if a migrating *task* does not interfere significantly with the execution time of other *tasks* executing on the same machine, why would it interfere with the programs of the local user, causing a performance loss? The answer is in the JVM. Even if the JVM is not executing heavy processing, it uses significant memory resources that can affect the local user, violating the InteGrade goal of transparency in terms of performance.

4 Experiments with the Simulation

We carried out a simulation - implemented in plain Java - with realistic parameters. In this simulation, the concept of *task group* is very important: a *task group* describes the twin *tasks*, grouping them together. Given *tasks* whose execution time is 60 min and the migration time is 5 min; 10 *tasks* of this kind are submitted to a small grid with 100 *machines*. Each *machine* has a probability p of dying each minute. Also, a dead *machine* has the same probability p of being turned on again. The search for dead *tasks* occurs every minute. We simulated *liveness degrees* $l \in \{2, 3, 4\}$ with different probabilities p. For each instance of (l, p), we made 1000 experiments. For each experiment we analyzed: (1) the average number of finished *task groups*, i.e. the number of *tasks groups* that finished execution among the 10 *task groups* and (2) the average execution time of the last *task group* that finished execution among the 10 *task groups*.

In Figure 2, we present Graph 1 showing the average number of *task groups* that have finished execution for a given probability p with *liveness degrees* of 2, 3 and 4. Graph 2 depicts the average execution time of the last *task group* that finished execution for a given probability p with *liveness degrees* of 2, 3 and 4. Graph 1 led us to two important conclusions: (1) $l = 4$ was the most reliable *liveness degree* in our experiments and (2) for high values of p, none of the analyzed values of l was reliable. Conclusion (1) is easy to explain, since the higher the number of *twins*, the less the probability of all the *twins* being killed. Conclusion (2) is explained by the fact that, even though many *machines* are turned on with higher values of p, many *machines* are killed also. With a high probability of *machines* being killed, we have a small probability for a given migration succeed. In fact, for a succeeded migration, neither the source nor the destination machine can be killed during the migration time y. Thus, the probability of a migration being succeeded is $(1 - p)^{2y}$.

By analyzing Graph 2, we can see that higher values of l led to higher execution times. We believe that this behavior happens because higher values of l imply more *tasks groups* that finish execution, since we have a higher number of *twins*. The migration processes used to keep the *twins* alive increase the execu-

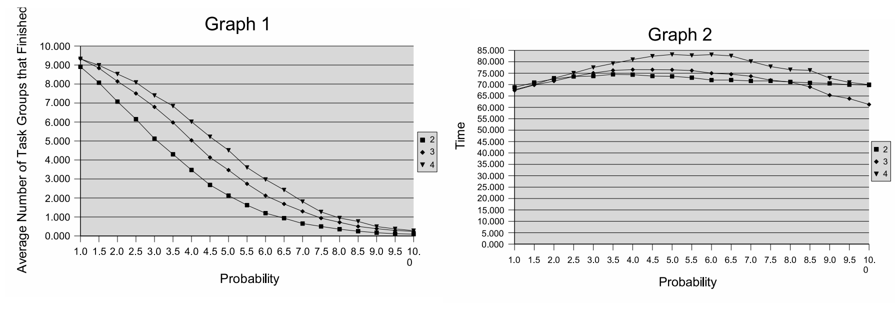

Fig. 2. Graph 1 shows the average number of *task groups* that finished execution. Graph 2 shows the average execution time of the last *task group* that finished execution. Probability $\times 10^{-2}$

tion time, which is reflected in the average. We can also see that the overhead (i.e. the time that exceeds the execution time of 60 min) may be significant. For instance, take $p = 6 \times 10^{-2}$ and $l = 4$: the overhead is about 36%.

5 Conclusion

We can see in the experiments that, for the given parameters, although higher *liveness degrees* were more reliable, they were not reliable enough for high probabilities. Also, we can see that, for high probabilities, the overhead may be significant, as there is an important influence of the migration process. We believe that a better reliability for high values of p could be reached by using higher values of l and having more *machines* available. With the given number of *machines*, higher values of l would have no effect, because they would cause more migrating *tasks* concurring for few *machines*.

References

1. Andrei Goldchleger, Fabio Kon, Alfredo Goldman, Marcelo Finger, and Germano Capistrano Bezerra. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience*, 16:449–459, March 2004.
2. Rodrigo M. Barbosa and Alfredo Goldman. Mobigrid - Framework for Mobile Agents on Computer Grid Environments. In *Mobility Aware Technologies and Applications (MATA 2004)*, pages 147–157. Springer-Verlag, 2005.
3. SETI@home. Site of the SETI@home project, 2004. `http://setiathome.ssl.berkeley.edu/`. Last visit on February, 2004.
4. Mitsuro Oshima and Guenter Karjoth. Aglets Specification 1.0. Technical report, IBM, may 1997. `http://www.research.ibm.com/trl/aglets/spec10.htm`.
5. Rocco Aversa, Beniamino Di Martino, Nicola Mazzocca, and Salvatore Venticinque. Mobile Agents for Distributed and Dynamically Balanced Optimization Applications. In *Proceedings of the 9th International Conference on High-Performance Computing and Networking*, pages 161–172. Springer-Verlag, 2001.