Quantifying the performance improvement of Migration in load sharing systems

Bertil FOLLIOT Yanal HAJMAHMOUD Pierre SENS
Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie
4, Place Jussieu, 75252 Cedex 05, FRANCE
e-mail : [yanal,folliot,sens]@src.lip6.fr

Abstract *Experimental and simulation studies have been realized to quantify the potential of migration to improve performance over the initial placement. In these studies, we tried to characterize the conditions under which this improvement can be obtained. We found that these conditions are strictly related to the heterogeneity of the task execution time and sometimes they are related to the number of hosts available and their relative speed. The various measures carried out show that the additional improvement is between* 15% *and* 30%, *and rarely attains* 35%. *In addition, we found that the placement algorithm is more adaptive to the applications and environments configuration than the migration algorithm.*

Keywords : Load Balancing, Process Allocation, Process Migration, Distributed Systems

1 Introduction

The dynamic load distribution systems can be classified into two categories according to whether they are equipped with a migration mechanism or they are limited to the initial placement. A system based on the initial placement transfers tasks from the local host to another at the time of their arrivals [1]. If the system provides a migration mechanism then a task can be transfered several times before being executed; or it can be suspended, be transfered to another host and re-takes its execution later [2]. So, task migration always involves transferring a partially executed task. This operation is generally expensive, since collecting task's state (which can be quite large) is often difficult. Douglis and Ousterhout [3] discuss various issues of preemptive task transfer.

Most past studies that have explicitly considered migration do not attempt to quantify its potential for improving performance, but simply assume that this potential is significant. A notable exception is the work of Leland and Ott [4], who show, using trace-driven simulation that migration can improve performance for the large jobs. Their results indicate an improvement of less than 25% for these jobs, however, they indicate no significant potential for improving the average performance over all jobs. Some analytical support for this negative result is provided by Kruskal and Weiss [5], but their models do not allow high variability in service demands, a situation in which the benefits of migration could be relatively significant.

Eager and al. [6] used a combination of analysis and simulation studies to examine the benefits offered by migration. They found that these benefits are too small compared with the benefits of the placement. Contrary of Eager and al., Harchol-Balter and Downey [7] found that the migration can add potential benefits to those of the initial placement. They derived a distribution that fits the lifetime of the UNIX processes and compared the two strategies for this distribution. However, they didn't take into account the impact of the tasks' arrival process on the performance.

In this paper, we presents our experience with GatoStar a real load sharing system focusing in the benefits of the migration mechanism added to the performance of the initial placement phase. Section 2 presents an overview of the GatoStar system. Section 3 presents some interesting results carried out by evaluating the performance of GatoStar. We conclude in Section 4.

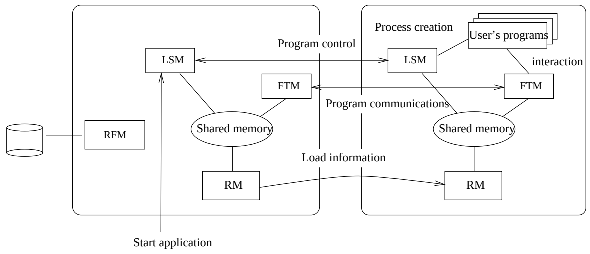

Figure 1: GatoStar architecture.

2 Overview of the GatoStar system

GatoStar is a fault tolerant load sharing facility [8], that was developed at the University of Paris 6. It is based on the implementation of two applications : Gatos and Star. Gatos is a load sharing manager which automatically distributes parallel applications among hosts according to multi-criteria algorithms. Star is a software fault manager which automatically recovers processes affected by host failures. GatoStar was built on top of Unix with no modifications to the UNIX kernel.

2.1 GatoStar Architecture

GatoStar consists of a ring of hosts which exchange information about hosts' functioning and processes' execution. Each host maintains a load vector which contains a view of all host loads. Periodically, each host sends its vector to its immediate successor. Load messages are also used to detect host failure.

The current prototype is composed of four daemons running on each host (Figure 1): (1) a load sharing manager (LSM) in charge of allocation and migration strategies, (2) a fault tolerance manager (FTM) responsible of checkpointing and message redirection, (3) a ring manager (RM) in charge of load transmission and failure detection, and (4) a replicated file manager (RFM) that implements the reliable storage.

To take into account the heterogeneous speed of hosts, the load of a host is defined as proportional to CPU utilization and inversely proportional to its CPU speed [9]. On each host, the load sharing manager locally computes an average load and uses failure detection messages to exchange load information.

An application is defined as a set of communicating processes connected by a precedence graph. The graph contains all qualitative information needed to execute the application and helpful for the load sharing manager: precedences and communications between processes, and files in use. To evaluate the need of each process, GatoStar provides a facility to keep track transparently of previous executions. Possibly, application designers may give directly appropriate indications.

The process allocation algorithms can be operated according to the following criteria: hosts load, process execution time, required memory, and communication between processes. In the application description, the programmer can specify appropriate allocation criteria for each process according to its execution needs. He/she specifies the set of hosts involved in the load distribution and provides different binary code versions for each process in order to deal with system heterogeneity.

A queuing network model of GatoStar has been developed to extend results with various applications and hardware configurations that are not currently available in our laboratory [10]. Our methodology consists of four main steps. In a first step, we develop a generic and configurable model. Then, in the configuration step, we specialize this model according to a static system configuration. In the tuning step, we evaluate performances of several applications and we compare these measurements with the real measures obtained from GatoStar. We refine possibly parameters of the specific model until to obtain simulated performances very close to real measurements. Once the simulator calibrated, we change heuristics of load balancing strategies and when a worthwhile optimization was found, this latter is integrated in GatoStar.

2.2 Placement

Allocating processes according to load allows to benefit from idle workstations. On each host, the load sharing manager locally computes an average load and uses failure detection messages to exchange load information. Thus, no extra communication is needed when allocating processes. The process is allocated to the most lightly loaded host, if the load of this host is below the *overload threshold*. Otherwise, the process is started on the originating host as all hosts in the system are heavily loaded, or used by interactive users. The main difficulty is to find a good value for the overload threshold. GatoStar uses adaptive values of thresholds that depends on the global load of the system.

2.3 Migration

The migration mechanism in GatoStar is periodically activated. The task transfer takes place when one of the three following events occur: a failure, an under-load, or overload detections.

GatoStar adopts a symmetrical scheme where the load sharing server periodically computes two conditions: (1) a faster workstation load is under the *reception threshold*, (2) the local host load is above the *migration threshold* and at least one workstation load is under the reception threshold. The former condition allows the potential use of fast available hosts, and the latter one allows to decrease the overload of the local station. If one or both conditions are true, the server starts the process selection step. In the process selection step, the local server chooses a process that has been executed locally for at least a given local execution time. In this way, It prevents short-lived process to be migrated and it also prevents a process from spending all its time in migration from one host to another.

The values of the migration and reception thresholds are dynamically adapted to the global load of the system.

Currently, GatoStar runs over 10 workstations (SUNs) linked by a local network (Ethernet of 10 Mb/s). Results mentioned below is a combination of real measures (until 10 hosts) and simulation measures (up to 10 hosts).

3 Performance measures

3.1 Overhead of the algorithms

In this section, we show the overhead of the placement and migration algorithms of the GatoStar system. For estimating this overhead, we executed an application of N tasks, each task is about 100 seconds, over a network of N hosts (one task by host). This estimation presents the overhead introduced by the task transfer during the initial placement and the overhead introduced to keep the balancing of the network. It does not contain the overhead introduced by the task transfers during migration process because it is difficult to derive the introduced overhead from the additional gain.

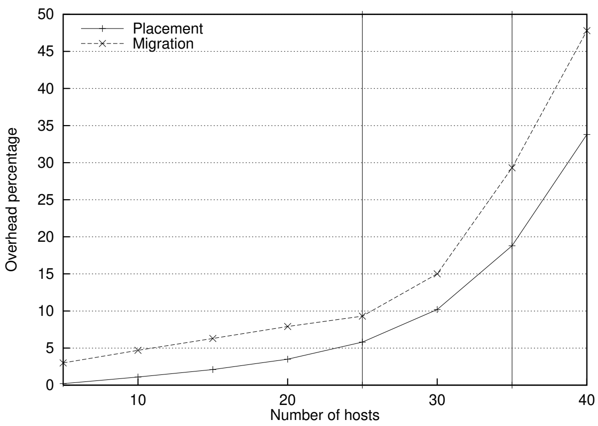

Figure 2: Overhead of placement and migration algorithms.

Figure 2 shows the overhead percentage of the execution time as a function of the hosts number. We see that the overhead is negligible if the number of hosts is less than 25. It is acceptable between 25 and 35 hosts. Upon 35, it becomes too high, and the system collapses. We observe that the overhead of the migration is always greater than that of the placement. The difference is about 3% for GatoStar.

3.2 Placement vs. migration

To investigate the additional benefits of the migration over the placement alone, we measured the performance of both algorithms for three types of applications : homogeneous applications whose tasks have quasi-identical execution time, heterogeneous applications whose tasks have execution time that fits a stochastic distribution, and finally irregular applications whose tasks have special execution time. In the following subsections, we give examples of these applications. We evaluated the efficiency of three strategies : placement alone, migration alone, and combined placement and migration.

Homogeneous applications

Figure 3 shows the speedup in case of an homogeneous application of 30 tasks. The execution time of each of task is 100 seconds. We see that the speedups due to the placement alone and to the placement-migration are in general semi-identical.

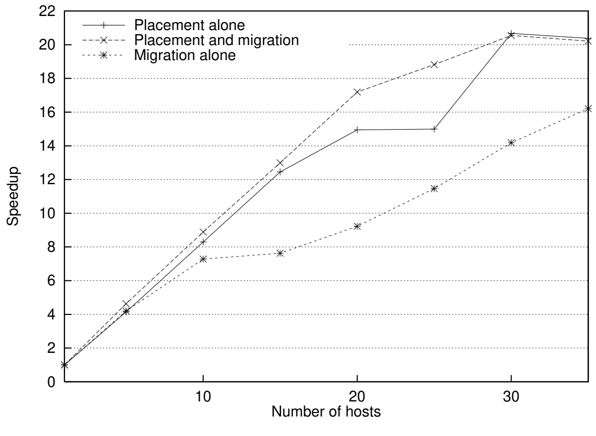

Figure 3: Homogeneous applications.

On the other hands, the speedup resulting from the migration is better between 15 and 30 hosts. This improvement is justified by the occurrence of unbalanced states during the execution. For example, for 20 hosts the initial placement is as follow : 2 tasks by host in the 10 fastest hosts and one task by host for the other 10 hosts. After an elapsed time, the last 10 tasks will finish their execution first. At this moment, the transfer of one of the two tasks present in a faster host towards a slower one (which is idle) will surely ameliorate the speedup.

The same reasoning is valid in case of 25 hosts. As shown in the curve, the additional speedup for 20 hosts is about 15% and for 25 hosts is about 26%. The performance degradation for the both strategies for 35 hosts is justified by the non-negligible overhead introduced by the load sharing algorithms.

Heterogeneous applications

The comparison between placement and migration is based on the execution time of an heterogeneous application of 500 tasks. We suppose that the task service time is a random variable of a given distribution. Many distributions have been tested: uniform, exponential, hyper-exponential, Gaussian, and the distribution proposed by Harchol-Balter [7].

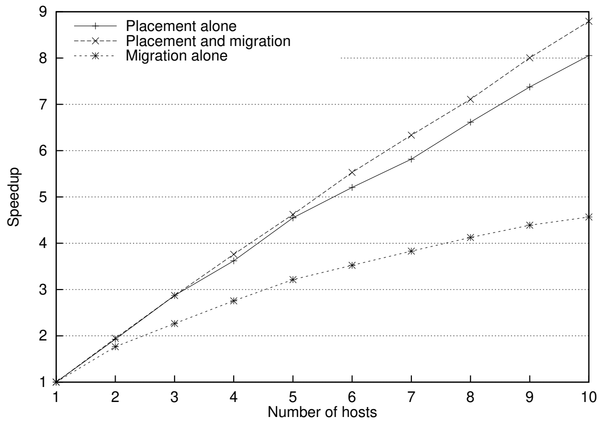

Figure 4: Heterogeneous applications.

For the majority of the tested distributions, the gain varies between 15 % and 35 %

(depending upon the distribution, its average value, and variation coefficient). Figure 4 shows one comparison for the hyper-exponential distribution with an average of 100 seconds. We see in the figure that the gain obtained by migration is always about 20 %. These results are in agreement with those of the pure simulation of Eager and al. [6].

Irregular applications

In order to clarify the importance of the migration in a load sharing system, we executed an irregular application of 30 tasks. The tasks are divided into three sets of identical size. They arrive in the following cyclic manner: one from the first set, one from the second, finally one from the third. Tasks of the first sets have an execution time of $\frac{100}{3}$ seconds, those of the second set have an execution time of $2\frac{100}{3}$ seconds, and those of the third set have a duration of 100 seconds.

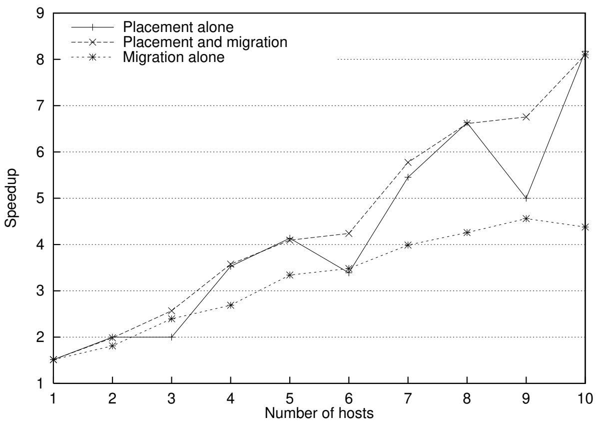

Figure 5: Irregular applications.

Figure 5 shows the execution time of this application with and without migration as a function of the hosts number. We see in this figure how the placement algorithm makes many bad decisions. These bad decisions are due to the big difference in the execution time of the tasks. So, the long tasks are grouped in the hosts numbered 3i. It is clear the migration avoids the erroneous choices and provides gain until 30 % in comparison with the placement.

3.3 Migration without placement

In the literature, there are some systems which use algorithms based only on the migration mechanism, without initial placement phase. To study the efficiency of the migration alone, we dis-activate the initial placement phase and re-execute the same applications seen above. Figures 3, 4 and 5 compare the speedup of the migration alone with the initial placement and the placement combined with migration. We observe the poor performance of the former strategy compared to the two others. The performance degradation is about 45% compared to the placement alone.

The bad performance of the migration algorithm, based on the dynamic thresholds, is due to the big inertia of this algorithm to re-act quickly for re-balancing the load of the system when a unbalance state is occurred. This inertia is necessary to avoid parasitic fluctuations on the system state. We conclude that this kind of algorithms is useful if they were used to supervise the placement algorithms and they can't be used as primary algorithms.

3.4 Impact of the application behavior

To study the adaptability of the placement and the migration mechanisms to the granularity of the applications, we supposed that a parallel application having a mass with D seconds (computation + I/O + communication) can be divided to N tasks of T seconds each. To simplify the problem, we suppose D is equal to N multiplied by T. We executed a parallel application of D equal to 1000 seconds on a system of 5 hosts. Figure 6 shows the speedup of the two algorithms as a function of N.

We see in Figure 6 the poor performance of the both algorithms for coarse-grain (for N less than 10). We justify that by the absence of the pseudo-parallelism inside each host. However, for the fine-grain we see only the migration algorithm has a poor performance while the speedup of the placement algorithm is almost constant when we increase N. The bad perfor-

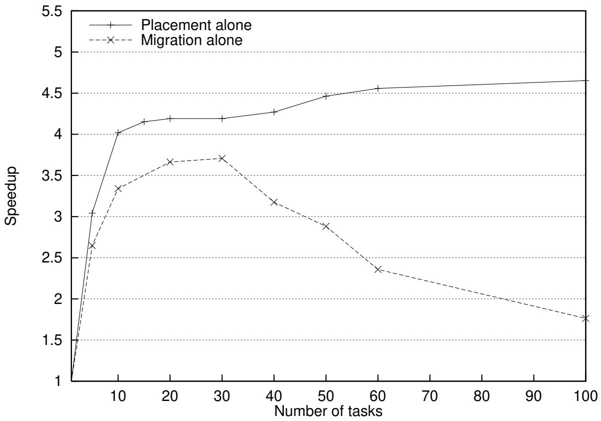

Figure 6: Granularity.

mance of the migration algorithm for the fine-grain is due to overhead produced and the gain obtained in migrating small tasks. In our case, when N is greater than 40 the speedup becomes very bad. We conclude that the migration algorithm is badly adaptive to the granularity of the applications.

3.5 Impact of the network latency

Figure 7 shows the speedup of the two algorithms as a function of the packet transfer time for an heterogeneous application of 50 tasks. The average execution time of a task is 100 seconds. The application is executed on a system of 5 hosts.

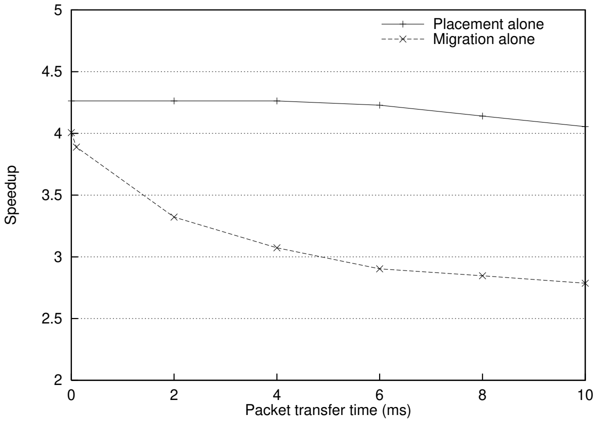

Figure 7: Packet transfer time.

From the figure we note that the impact of the network speed on the performance of the migration is more important than that of the placement, this is because the number of task transfers in the second one is negligible relative to the former one. But in the two cases, the performance degradation is not so big because the overhead of the tasks transfer is negligible relative to the global execution time of the application.

3.6 Impact of the heterogeneity

To study the adaptability of the placement and the migration mechanisms to the speed heterogeneity of the hosts, we measured the performance of the two mechanisms for eight network configurations. The eight networks have the same overall computation capacity which we defined it by the sum of the speed of network hosts. The first network is homogeneous. The second one is our laboratory network. The others are heterogeneous networks randomly chosen. Figures 8 shows the speedup of the two algorithms for an heterogeneous application of 50 tasks executed on the eight networks. The average execution time of a task is 100 seconds.

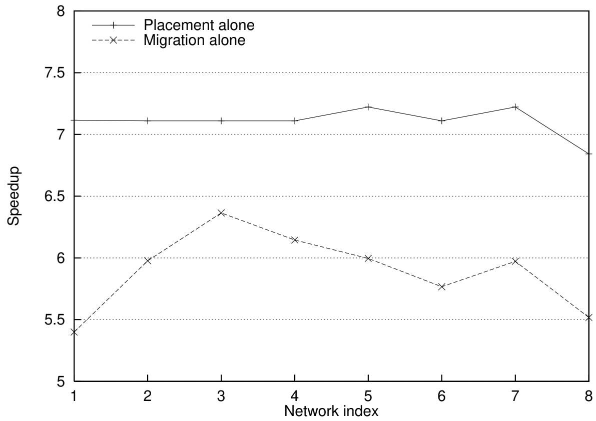

Figure 8: Eight equivalent networks.

From the figure we note that the speedup of the placement is almost constant for the eight configurations contrary to the speedup of the migration which varies form one configuration to another. We conclude that the placement is more adaptive to the network configuration than the migration.

4 Conclusion

In this paper, we presented the performance of the placement and migration algorithms of the GatoStar system. We showed that the static overhead of the migration (introduced by supervising the load balancing of the system without any task transfers) is about 3% greater than placement.

We observed that both homogeneous and heterogeneous applications can lead to the occurrence of unbalanced states for which the activation of the migration is necessary to rebalance the load. The migration in this case is surely profitable. However, the maximum additional improvement observed is about 35%. We showed also that the migration alone can never be efficient as the placement alone, so we believe of the dominant role of the placement and the auxiliary role of the migration in a load sharing system.

Finally, we showed the strong dependency of the migration on the applications behavior and the network characteristics. We concluded that the placement is more stable than the migration.

References

[1] N. G. Shivaratri, P. Krueger, and M. Singhal. Load distributing for locally distributed systems. *Computer Journal*, 25:33–44, 1992.

[2] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. Technical report, The Open Group Research Institute, 1996.

[3] F. Douglis and J. Ousterhout. Transparent process migration : Desing alternatives and the sprite implementation. *Software-Practice and Experience*, 21(8), 1991.

[4] W. Leland and T. Ott. Load balancing heuristics and process behavior. *ACM Performance Evaluation*, 14:54–69, 1986.

[5] C. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. *IEEE Transactions on Software Engineering*, SE-11(10):1001–1016, 1985.

[6] D. L. Eager, E. D. Lazowska, and J. Zahorjan. The limited performance benefits of migrating active processes for load sharing. In *ACM Sigmetrics Conference on Measuring and Modeling of Computer Systems*, pages 24–27, 1988.

[7] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3), 1997.

[8] B. Folliot and P. Sens. Gatostar : A fault tolerant load sharing facility for parallel applications. In *Proc. First European Dependable Computing Conf.*, pages 581–598, 1994.

[9] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software-practice and experience*, 23(12):1305–1336, 1993.

[10] Y. Hajmahmoud and B. Folliot. Queuing network model for load balancing evaluation in a heterogeneous distributed system. In *The 1998 International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 4, 1998.