Performance Evaluation of a Load Sharing System on a Cluster of Workstations

Yanal Hajmahmoud, Pierre Sens, and Bertil Folliot

Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie
4, Place Jussieu, 75252 Cedex 05, FRANCE
[yanal,sens,folliot]@src.lip6.fr

Abstract. This paper presents a methodology to evaluate the performance of load allocating algorithms using queuing network facilities. First, we develop a generic model of load sharing systems. Then, a specific model is derived from the generic model that simulates a real load sharing system. This phase consists of successive manipulations of algorithms parameters until reaching satisfied approximations. Once the suitable parameters have been tuned, which configure the specific model, it is easy to evaluate the performance of the load allocating algorithms for various applications and hardware configurations.

1 Introduction

To maximize the performance in a distributed-computing environment, it is essential to evenly distribute the load among the hosts. Many researchers have demonstrated that simple load sharing algorithms produce considerable performance improvement [4]. Classical algorithms are *CPU load balancing* where processes are transferred across the network from highly loaded host to a lower loaded host. In multi-criteria approaches, some heuristics propose to take into account both application behaviors and the global state of the system [1, 2]. The main difficulty of such algorithms is to find the appropriate parameters geared to each application such as load thresholds or the weight of various criteria.

We propose a configurable model to help the design of process allocation algorithm. In a first step, we develop a generic model of load sharing systems. 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 an existing migration system. We refine parameters of the specific model until getting simulated performances that are very close to real measurements. Once the simulator is calibrated, we change heuristics of load balancing strategies and when a worthwhile optimization is found, the latter is integrated in the real system.

The remainder of this paper is organized as follows. The next section outlines common load sharing policies and presents the GatoStar system which we have chosen to tune our simulator. The Section 3 describes the model. Finally, the Section 4 presents our performance evaluation study.

2 The Load Sharing System

GatoStar is a load sharing facility [2], that we have developed previously at the University of Paris 6. It automatically distributes parallel applications among hosts according to multi-criteria algorithms. A migration mechanism is periodically activated to supervise the load balancing in the network.

GatoStar was built on top of Unix with no modifications to the UNIX kernel. It works on a set of heterogeneous workstations connected by a local area network. GatoStar defines migration domains of compatible hosts, where processes can freely move. The basic architecture consists of a ring of hosts which exchange information about hosts' functioning and processes' execution.

To take into account the heterogeneous features of hosts, the load of a host is defined as directly proportional to CPU utilization and inversely proportional to its CPU speed [5]. On each host, the load sharing manager locally computes an average load and uses the ring to exchange load information.

Processes are allocated to the most lightly loaded host, only if the load of this host is below the *overload threshold*. Otherwise, a process is started on its originating host as all hosts in the system are highly loaded, or used by interactive users.

To react to applications and environment evolution, GatoStar includes a process migration mechanism. Environment evolution results from foreign activities : an interactive user logs in or sends a remote job to an already loaded host. Application evolution stems from variations in resource usage (processor occupation, communication between processes, and memory demands).

The migration decision is based on two opposite load thresholds : *migration and reception thresholds*. The load sharing manager periodically verifies 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 host. If one or both conditions are true, the manager of the loaded host chooses a process that has been executed locally for at least a given slice time. This prevents short-lived process to be migrated and also prevents process thrashing.

3 The Model

The developed model is a network of *service stations*, constructed using the QNAP2 package utility. We have distinguished four principal types of activities inside each host of the system. The first activity concerns the injection of programs by the host's owner. The second one concerns the distribution of processes derived from the user programs among the network. The third one concerns the execution of the processes at the host. The last one concerns the interface between the host and the network. We have gathered these activities into separated modules. Therefore, each module has a well known functionality which is defined by a set of algorithms with their appropriate parameters. The generic property of the model comes from the implementation of several algorithms inside each model. At the configuration phase, the programmer chooses his/her algorithms and the values of their parameters. For example, the programmer can choose the load sharing algorithms (cyclic, random, specific algorithms) for the load distribution module.

A host of the system is described by five modules: a *program generator* (that generates parallel applications and sequential programs simulating the utilization of the host by his/her owner), a *local clock*, a *distributing manager* (that simulates the load sharing algorithms), an *executing manager* (that simulates the processors, disks, memory and communication operations) and finally, a *communicating manager* (that simulates the interface between the site and the network). The LAN is represented by one queue with one or several servers. The queue length reflects the load in the network. The number of servers reflects the accessibility of the network by the hosts.

We generate sequential programs, and parallel applications. A sequential program consists of one task which is executed in the local host and has no interaction and communication with the other tasks of the host. This type of programs (called *local tasks*) simulates the commands launched by the local user of the workstation (ls, lpr, etc). The execution time of a local task is divided in *CPU-segment* and *I/O-segment*. The CPU-segment represents the execution time on the processor and the I/O-segment represents the I/O operations on the disk. The generation rate, the execution time and the ratio between the CPU-segment, and the I/O-segment are parameters.

A parallel application is composed of several tasks (called *parallel tasks*) which can be executed in parallel. Parallel applications are defined by their precedence graphs. A parallel task is composed, in addition to the *CPU-segment* and the *I/O-segment*, of a *Comm-segment* which simulates the communication with the other tasks of the application. The user determines the sizes of the segments.

To tune the parameters of this generic model, we have performed successive manipulations of the model parameters until reaching results almost similar to those obtained by the real load sharing system. We have compared many executions on both our simulator and the real system including computation, I/O and communication operations. The measures have been carried out for a network of 8 heterogeneous hosts equipped with GatoStar. We have measured that the difference between the two sets of measures is less then 10%.

4 Performance study

In this section, we study the impact of load sharing strategies and their parameters on the system performance. We concentrate this study on the values of the load thresholds, the load balancing policies, and the weight of I/O operations compared to communications over a network.

4.1 Thresholds values

Thresholds are key features and directly influence the performance of load balancing strategies. We have compared two policies of the thresholds. In the classical one, which is implemented in GatoStar, allocation and migration decisions are based on fixed thresholds. In the adaptive one, thresholds depend on the global system state (the current global load and the hosts characteristics).

Figure 1 compares the performance of these two policies. We have executed two parallel applications : application A is composed of 40 tasks that have identical execution times, application B is composed of 48 tasks that have different execution times (24 short tasks and 24 long tasks). We measured the speedup compared to a local execution for the two policies. We note that the adaptive strategy always improves the performance of the network, particularly for heterogenous application (up to 30% for application B).

This study has encouraged us to implement the adaptive strategy in the GatoStar system. Similar remarkable benefits have been observed in the real environment.

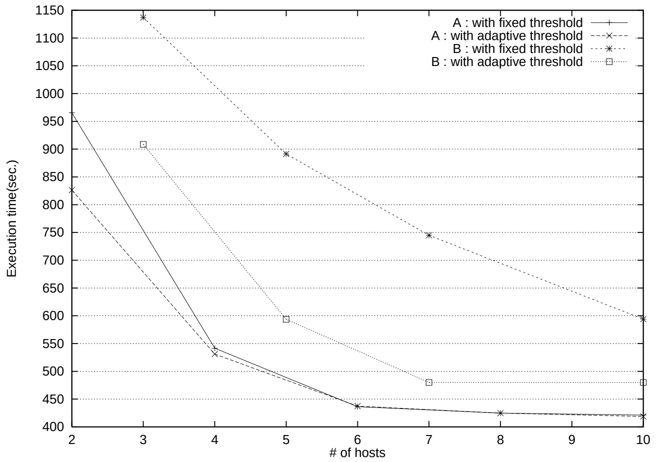

Fig. 1. Adaptive thresholds.

4.2 Placement vs. Migration

To investigate the additional benefits of the migration over the initial placement, we have compared four load sharing strategies : two adaptive load sharing strategies (placement with and without migration), and two static strategies (random and cyclic which is the PVM load sharing algorithm).

Figure 2 shows the speedup of the four strategies for a parallel application composed of 30 identical tasks as a function of the hosts number. We observe that the adaptive load sharing strategies improves the performance over static approaches (as random and cyclic). Moreover, migration balances the load more evenly than placement.

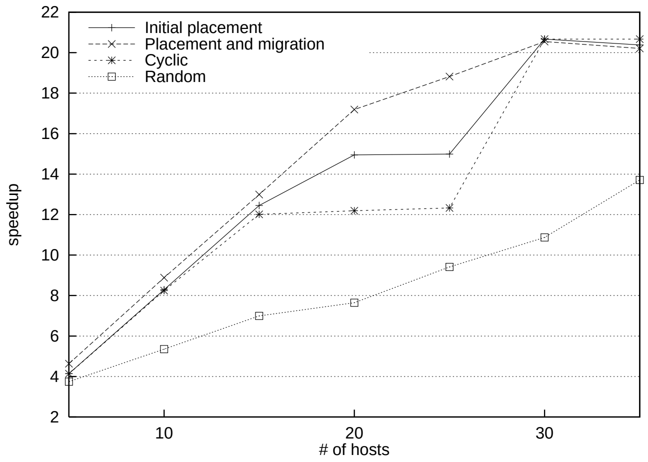

Fig. 2. Comparison of load sharing policies.

For the majority of the tested applications, we found that the additional gain of the migration over the initial placement varies between 15 % and 35 % (depending on the application and the network configuration). Therefore, the combination of both strategies is particularly important.

4.3 Impact of Communication and I/O

The majority of the studies of the load balancing problem consider applications using only the CPU resources. Few studies show the influence of communication or I/O on the overall performance [3]. To investigate this influence, we have executed two parallel applications composed of 20 tasks, on a network of 10 hosts. Each task of the first application contains *CPU-segment* and *I/O-segment*. Each task of the second application contains *CPU-segment* and *Comm-segment*. Figure 3 shows the speed-up as a function of the ratio of communication over computation for the first application and the ratio of I/O over computation for the second application.

We note that the first application has a higher speed-up than the second one. Therefore, the communication has more important impact on the performance than I/O impact. Consequently, the communication criterion must have a higher priority than the I/O in a multi-criteria load sharing heuristic.

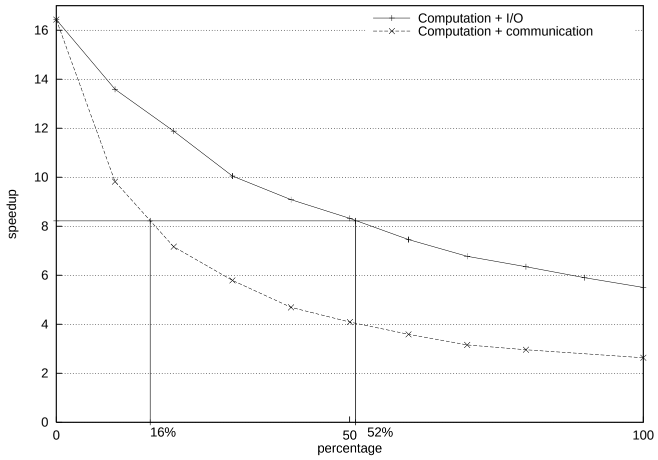

Fig. 3. Communication and I/O.

5 Conclusions

We presented a methodology to evaluate the performance of load sharing algorithms. We began by designing a generic model of load sharing systems, then we derived a specific model that mimics a real system.

Among many results obtained, we highlighted the importance of the adaptive thresholds policy against the fixed thresholds policy. We measured the additional benefits of the migration mechanism over the initial placement. However, the maximum additional amelioration observed is about 35%. We also concluded that the communication operations have more important impact on the overall performance than I/O operations.

References

1. A. Barak and O. La'adan. The mosix multicomputer operating system for high perfromance cluster computing. *Journal of Future Generation Computer Systems*, 13(4), 1999.
2. P. Sens and B. Folliot. Gatostar : A fault tolerant load sharing facility for parallel applications. In *Proc. First European Dependable Computing Conf.*, pages 581–598, 1994.
3. J. P. Singh, E. Rothberg, and A. Gupta. Modeling communication in parallel algorithms: A fruitful interaction between theory and systems? In *In Proceedings of the 6th annual ACM symposium on parallel algorithms and architectures*, 1994.
4. Y. T. Wang and J. T. Morris. Load sharing in distributed systems. *IEEE Transactions. on Computers*, c-34:204–217, 1985.
5. 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.