THE DESIGN OF LOAD BALANCING STRATEGIES FOR DISTRIBUTED SYSTEMS

Rafael Alonso

CS-TR-035-86

May, 1986

FUTURE DIRECTIONS IN COMPUTER ARCHITECTURE AND SOFTWARE WORKSHOP
May 5-7, 1986. Seabrook Island, S.C.

THE DESIGN OF LOAD BALANCING STRATEGIES FOR DISTRIBUTED SYSTEMS

Rafael Alonso

Department of Computer Science
Princeton University
Princeton, N.J. 08544
(609) 452-3869

ABSTRACT

In this paper we consider the problem of designing and selecting load balancing mechanisms for distributed systems based on local area networks. In particular, we will focus on the type of information needed to make balancing decisions and on the desirable properties of the decision algorithms. We also describe our current approach to this problem, which essentially consists of carrying out a series of experiments on a prototype load balancing implementation. Finally, we present the insights we have derived from our experimental results.

May 8, 1986

THE DESIGN OF LOAD BALANCING STRATEGIES FOR DISTRIBUTED SYSTEMS

Rafael Alonso

Department of Computer Science
Princeton University
Princeton, N.J. 08544
(609) 452-3869

1. Introduction

Many of the currently existing computing environments consist of a heterogeneous collection of of workstations and mainframes connected by a high bandwidth local area network (LAN). One of the main benefits of working in a distributed system created for such an environment is being able to share scarce resources with other users of the network; but one resource that is often not shared is the processing capacity of the network nodes. In many systems, the scheduling of user jobs is individually carried out by each processor, and the computations of the users logged on at any one machine are performed locally. The decentralization of CPU management, coupled with large differences in the numbers (and types) of users connected to each of the nodes in the network, often can lead to situations where there are great disparities in load among the machines on the network. For example, at our local computer center, as the due date for a class assignment approaches, the processor assigned to the students in that class becomes heavily loaded, while other machines are underutilized. Although users can determine by themselves that an imbalance exists, and remotely log onto another computer, we feel that, in order to prevent a chaotic situation and to aid naive users, it is best to develop strategies that can solve the load balancing problem in an automatic way, much in the way that users now depend on virtual memory techniques to manage their memory space.

We are currently exploring the design of load balancing strategies for LAN-based distributed systems. In this paper, we describe some of the strategies that we are considering for implementation, as well as the criteria we are using to determine which schemes are appropriate for a given environment.

In the next section we provide some details concerning our approach to the problem. Sections 3 and 4 deal with load metrics and decision policies respectively (these terms are defined in Section 2). The last section of the paper discusses the current state of our work and presents some preliminary conclusions.

This research partially supported by New Jersey Governor's Commission Award No. 85-990660-6

2. Our Approach

Although load strategies have been studied in the past, most of this work has been carried out within a theoretical framework (see [Chu1980] for a survey of some of these approaches); moreover, some of the researchers carrying out those studies have made simplifying assumptions that may not hold in practice. Also, there have been a small number of load balancing implementations (for example, see[Hwang1982]), but, as far as we know, the strategies employed were chosen without extensive study. Our work differs from most of the previous research in that, while we are interested in actually implementing a load balancing mechanism, we will not chose a strategy in an *ad hoc* fashion, but rather, we plan to devote extensive study to the choice of strategy to be programmed.

Before we describe our approach further, a few remarks are needed. A load balancing strategy is composed of two parts, a *load information* aspect (i.e., what information will be used to determine the load in the machines of the network), and a *decision policy* (i.e., given the load information, how will it be used to decide where to run a job). In order to speak meaningfully of a machine being "more loaded" than another, we define a *load metric* as a real-valued function of the load information. For example, the load information could be composed of the number of processes in the CPU ready queue for a given interval; a possible load metric would then be the average number of ready jobs during the last minute. This metric is essentially the "load average" metric provided by the UNIX 4.2 BSD [Leffler1984] **uptime** command. In the next two sections we consider load metrics and decision policies in greater depth.

Our current study of the load balancing problem comprises three phases. First, to consider what are suitable load metrics for our environment. Then, to list a number of possible decision policies that make use of the load metrics being studied. Finally, implementing various combinations of policies and metrics, in order to study their performance for a number of synthetic workloads.

We have already developed a prototype implementation of a load balancing mechanism, which can be used with different load strategies. It consists essentially of a shell that can make scheduling decisions based on load information broadcast by cooperating daemons (see [Alonso1986a] for more details). Our prototype now runs in a laboratory consisting of a variety of SUN workstations, connected by an Ethernet [Metcalfe1976]. Although experiments are still being carried out, we will briefly comment on our current results in the concluding section of the paper.

3. Load Metrics

The selection of a load metric requires a careful definition of what is meant by the load of a processor. It seems clear that load should be defined, at least partially, in terms of a set of performance indices (such as CPU utilization or mean number of I/O requests), but it is less clear that two different processes

should use the same definition of load; for example, an I/O intensive job will probably perceive load in a different way than a CPU-bound job. Whether the benefits, if any, of using different load metrics for different tasks are sizable is not immediately obvious; at any rate, at present we are considering only global load metrics. Our rationale for this decision is twofold. In the first place, in our computational environment we typically do not know the characteristics of the jobs being executed, and thus, cannot compute job-specific load metrics. Secondly, we feel that a simpler technique has a better chance of working well in an implementation than a more complex approach.

A compromise between global and job-specific load metrics is to compute a different load metrics for each job class. The simplest such scheme would involve categorizing processes as either CPU or I/O bound. Although promising, this approach would not be of interest in our system, since most user jobs obtain their data (more precisely, all their pages) from a network file server. Thus, there is little one can do (in terms of migrating the task) for I/O bound jobs, except to exclude them from the load balancing algorithm. (Actually, this is equivalent to a load metric that has the same value for all machines under all conditions.) More will be said about the types of jobs that should not be migrated in the next section.

Another important issue is that, since load metric information cannot be constantly broadcast (because it would be too costly), the rate at which such broadcasts are made needs to be studied. The problem that arises is that machines are making their job scheduling decisions based on possibly stale data. This could lead to unstable behavior; for example, an idle node could start receiving migrated jobs from many other machines, and by the time it broadcast that it was overloaded, perhaps too many jobs would have been sent to it. We have already seen such behavior in our previous study of the possible improvements of introducing load balancing strategies in distributed database query optimizers [Alonso1986b].

Actually, the cost of frequent load information broadcasts involves two factors: the cost of computing the load metric, and the overhead of sending and receiving the messages over the network. One possible compromise would be to gather the load information more frequently, but broadcast it at less frequent intervals. The rationale for this is that, with this approach, each machine has a better idea of its local load, and can start migrating all of its jobs when it is overloaded. One of the results described in [Alonso1986b] suggests that, at least in the context of database jobs, this strategy can be quite successful. We plan to determine if that result holds for general purpose jobs.

A possible candidate for a load metric is the load average described above. Another is the *effective load average*, defined as the load average divided by the processor MIPS rate. A third is CPU utilization. The number of ready jobs plus the number of disks requests per unit of time seems a more comprehensive measure of load than a metric that involves solely the demand on the CPU. Another possibility is to compute the utilization of the bottleneck resource at every machine (since the throughput of a system is dominated by the scarcest

resource, the utilization of this critical resource is a good indication of the node load). Lastly, a binary metric ("idle" or "not_idle") would be an appropriate choice if we were interested in migrating jobs primarily to unused machines on the network. Currently, we are planning to study most of the metrics just described.

The are a number of issues involved in evaluating a load metric:

[1] The **stability** of a metric is important, because if it responds too quickly to minor changes in system parameters it may lead to unstability in our load balancing strategies. The measurement technique used to compute the metric is relevant here (for example, exponentially smoothing a sample metric can improve the stability of the metric).

[2] We are also interested in the **generalizability** of the metric, since we desire our schemes to be useful for non-UNIX systems as well as being valid for a heterogeneous set of hardware configurations.

[3] Clearly, the **implementability** of the metric is of primary concern. Since we intend to carry out experiments using the various metrics, the ones that are either too difficult or too expensive to implement or to compute must be discarded.

[4] Finally, there must be **empirical evidence** that the metric to be implemented actually reflects our intuitive notion of "load". This can be determined by experimental study.

Before we leave this topic it should be pointed out that, clearly, a load metric is only as good as the operating system statistics on which it is based. In the case of UNIX 4.2 BSD, it is not clear that the statistics provided by the kernel can be trusted completely. Thus, we also plan to carry out a study of the statistics gathering software of BSD UNIX.

4. Decision Policies

Before we describe some of the decision policies that seem attractive, we will touch upon some of the salient features of such policies. Policies can be categorized as more static or more dynamic; for example, always using the same computational server to offload a machine would be a very static policy, while choosing the server at runtime is more dynamic. Some policies require knowing varying amounts of information about a process in order to schedule it (e.g., a strategy may require that we know if a process is CPU or I/O bound). The scheduling decision may be made at a central site or in a distributed fashion (we are only interested in the latter); also, the location of the decision mechanism may impact the number of messages that have to be sent to communicate load information. A key feature of any policy is whether it allows preemption or not (i.e., whether we are allowed to keep migrating jobs after they start executing); we will not consider such policies since we cannot currently implement them in a UNIX environment. Some policies may limit the processors that may be chosen; a possible strategy may be to let only idle nodes compete for tasks, or to consider only machines with a certain MIPS rate for task allocation. The

choice of sender-initiated versus receiver-initiated balancing must be settled (i.e., whether to allow busy nodes to look for an idle node, or to have idle nodes advertise their ability to work). Finally, it is clear that some jobs should never be migrated; this may be so for a variety of reasons: perhaps the jobs should only be run locally for security reasons, or maybe the computational demands of the task are so slight that the overhead of moving it overshadows any possible performance gain. There are still many other issues to be considered: do we optimize for the current task or do we look at sets of tasks? do we examine only jobs that have arrived at our local decision maker mechanism or at all the incoming jobs in the network? do we remember previous task assignments when we make a decision? which performance index do we focus on?

There are many policies that could be of practical use. Perhaps the most natural policy is to send jobs to the "least loaded" machine. A variation that may prove more stable consists of migrating jobs to any machine whose load is less than the local load by a specified amount. In order to decrease the chance that all busy machines select the same host to which they will migrate their tasks, each machine could choose the destination randomly from the n least loaded processors (for some n). Another alternative is to only move jobs to idle processors. Finally, always sending the jobs of overloaded processors to a powerful computational server may be reasonable for some environments.

In judging a decision policy we may consider a number of factors:

[1] As we mentioned above, the **stability** of the policy under imperfect information is important, since we expect that imperfect information will be the rule rather than the exception.

[2] We would prefer that the **cost** of the load balancing scheme be negligible compared to its benefits (note that there are two types of costs, the costs to users of using the system, and the cost to non-users of the added overhead of running the decision mechanism).

[3] Also, the load balancing scheme must not force all the processors in the network to guarantee a given level of service; in particular, some processors may refuse service or allow only certain classes of tasks to be assigned to them. Thus, the amount of **autonomy** that machines have under a given policy is important. (Note that the receiver-initiated policies mentioned above seem to simplify achieving this goal.)

[4] Finally, the amount of **transparency** in the load balancing scheme is also of critical importance, since it would be desirable to maintain users unaware of the fact that their jobs may be running remotely.

5. Conclusions

In this paper we have presented our ideas concerning the implementation of load balancing mechanisms. We have pointed out possible load metrics as well as sample decision policies, and have detailed some of the relevant issues involved in making a choice from the many possible strategies. It seems clear to us that the selection of a load balancing strategy involves many complex choices

which require careful study, and it is certainly not clear *a priori* which strategy to pursue.

At the present time, we are actively involved in a series of experiments that explore some of the ideas presented in this paper. Our previous work ([Alonso1986a] and [Alonso1986b]) suggests that the gains due to load balancing can be quite sizable, and our current results continue to be promising. Moreover, we feel that this area of research contains many problems of both practical and research interest and merit further study.

References

Alonso1986a.

Alonso, Rafael, Goldman, Phillip, and Potrebic, Peter, "A Load Balancing Implementation for a Local Area Network of Workstations," *Proceedings of the IEEE Workstation Technology and Systems Conference*, March 18-20, 1986.

Alonso1986b.

Alonso, Rafael, "Query Optimization in Distributed Database Systems Through Load Balancing," Ph.D. Dissertation, U.C. Berkeley, 1986.

Chu1980.

Chu, W. W., Holloway, L. J., Lan, M., and Efe, K., "Task Allocation in Distributed Data Processing," *IEEE Computer*, November 1980.

Hwang1982.

Hwang, K., Croft, W. J., Goble, G. H., Wah, B. W., Briggs, F. A., Simmons, W. R., and Coates, C. L., "Unix Networking and Load Balancing on Multi-Minicomputers for Distr. Proc.," *IEEE Computer*, April 1982.

Leffler1984.

Leffler, S., Joy, W., and McKusick, K., *4.2 BSD System Manual,* Computer Systems Research Group, University of California, Berkeley, 1984.

Metcalfe1976.

Metcalfe, R. M. and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks," *CACM*, vol. 19,7, pp. 395-404, July 1976.