Interpreting Stale Load Information *

Michael Dahlin
Department of Computer Sciences
University of Texas at Austin
dahlin@cs.utexas.edu

UTCS Technical Report TR98-20

Abstract

In this paper we examine the problem of balancing load in a large-scale distributed system when information about server loads may be stale. It is well known that sending each request to the machine with the apparent lowest load can behave badly in such systems, yet this technique is common in practice. Other systems use round-robin or random selection algorithms that entirely ignore load information or that only use a small subset of the load information. Rather than risk extremely bad performance on one hand or ignore the chance to use load information to improve performance on the other, we develop strategies that interpret load information based on its age. Through simulation, we examine several simple algorithms that use such load interpretation strategies under a range of workloads. Our experiments suggest that by properly interpreting load information, systems can (1) match the performance of the most aggressive algorithms when load information is fresh, (2) outperform current algorithms by as much as 60% when information is moderately old, (3) significantly outperform random load distribution when information is older still, and (4) avoid pathological behavior even when information is extremely old.

1 Introduction

When balancing load in a distributed system, it is well known that the strategy of sending each request to the least-loaded machine can behave badly if load information is old [11, 18, 21]. In such systems a "herd effect" often develops, and machines that appear to be underutilized quickly become overloaded because everyone sends their requests to those machines until new load information is propagated. To combat this problem, some systems

*This work was supported in part by an NSF CISE grant (CDA-9624082) and grants Novell and Sun. Dahlin was also supported by an NSF CAREER grant (9733842).

adopt randomized strategies that ignore load information or that only use a small subset of load information, but these systems may give up the opportunity to avoid heavily loaded machines.

Load balancing with stale information is becoming an increasingly important problem for distributed operating systems. Many recent experimental operating systems have included process migration facilities [2, 6, 9, 16, 17, 23, 24, 25, 26, 30] and it is now common for workstation clusters to include production load sharing programs such as LSF [31] or DQS [10]. In addition, many network DNS servers, routers, and switches include the ability to multiplex incoming requests among equivalent servers [1, 5, 8], and several run-time systems for distributed parallel computing on clusters or metacomputers include modules to balance requests among nodes [12, 14]. Server load may also be combined with locality information for wide area network (WAN) information systems such as selecting an HTTP server or cache [13, 22, 28]. As such systems include larger numbers of nodes or the distance between nodes increases, it becomes more expensive to distribute up-to-date load information. Thus, it is important for such systems to make the best use of old information.

This paper attempts to systematically develop algorithms for using old information. The core idea is to use not only each server's last reported load information (L_i), but also to use the age of that information (T) and an estimate of the rate at which new jobs arrive to change that information (λ). For example, under a *periodic update* model of load information [21] that updates server load information every T seconds, clients using our algorithm calculate the fraction of requests they should send to each server in order to equalize the load across servers by the end of the epoch. Then, for each new request during an epoch, clients randomly choose a server according to these probabilities.

In this paper, we devise load interpretation (LI) algorithms by analyzing the relevant queuing systems. We then evaluate these algorithms via simulation under a range of load information models and workloads. For our LI algorithms, if load information is fresh (e.g., T or λ or both are small), then the algorithms tend to send requests to machines that recently reported low load, and the algorithms match the performance of aggressive algorithms while exceeding the performance of algorithms that use random subsets of load information or pure random algorithms that use no load information at all. Conversely, if load information is stale, the LI algorithms tend to distribute jobs uniformly across servers and thus perform as well as randomized algorithms and dramatically better than algorithms that naively use load information. Finally, for load information of modest age, the LI algorithms outperform current alternatives by as much 60%.

Other algorithms that attempt to cope with stale load information, such as those proposed by Mitzenmacher [21], have the added benefit that by restricting the amount of load information that clients may consider when dispatching jobs, they may reduce the amount of load information that must be sent across the network. We examine variations of the LI algorithms that base their decisions on similarly reduced information. We conclude that even with severely restricted information, the algorithms that use LI can outperform those that do not. Furthermore, modest amounts of load information allow the LI algorithms to

achieve nearly their full performance. Thus, LI decouples the question of how much load information should be used from the question of how to interpret that information.

The primary disadvantage of our approach is that it requires clients to estimate or be told the job arrival rate (λ) and the age of load information (T). If this information is not available, or if clients misestimate these values, our algorithms can have poor performance. We note, however, that although other algorithms that make use of stale load information do not explicitly track these factors, those algorithm do implicitly assume that these parameters fall within the range of values for which load information can be considered "fresh;" if the parameters fall outside of this range, those algorithms can perform quite badly. Conversely, because our algorithms explicitly include these parameters, they gracefully degrade as information becomes relatively more stale.

The rest of this paper proceeds as follows. Section 2 describes related work with a particular emphasis on Mitzenmacher's recent study [21], on which we base much of our methodology and several of our system models. Section 3 introduces our models of old information and Section 4 describes the load interpretation algorithms we use. Section 5 contains our experimental evaluation of the algorithms, and Section 6 summarizes our conclusions.

2 Related work

Awerbuch et. al [3] examined load balancing with very limited information. However, their model differs considerably from ours. In particular, they focus on the task of selecting a good server for a job when other jobs are placed by an adversary. In our model, jobs are placed by entities that act in their own best interest but that do not seek to interfere with one another. This difference allows us to more aggressively use past information to predict the future.

A number of theoretical studies [4, 7, 15, 20, 27] have suggested that load balancing algorithms can often be quite effective even if the amount of information examined is severely restricted. We explore how to combine this idea with LI in Section 5.6.

Several studies have examined the behavior of load balancing with old or limited information in queuing studies. Eager et. al [11] found that simple strategies using limited information worked well. Mirchandaney et. al [18, 19] found that as delay increases, random assignment performs as well as strategies that use load information.

Several system have used the heuristic of weighing recent information more heavily than old information. For example, the Smart Clients prototype [29] distributed network requests across a group of servers using such a heuristic. Additionally, a common technique in process migration facilities is to use an exponentially decaying average for to estimate load on a machine (e.g., $Load_{new} = Load_{old} * k + Load_{current} * (1 - k)$ for some $k < 1$). Unfortunately, the algorithms used by these systems are somewhat *ad hoc* and it is not clear under what circumstances to use these algorithms or how to set their constants. A goal of our study is

to construct a systematic framework for using old load information.

Our study most closely resembles Mitzenmacher's work [21]. Mitzenmacher examined a system in which arriving jobs are sent to one of several servers based on stale information about the servers' loads. The goal in such a system is to minimize average response time. He examined a family of algorithms that make each server choice from small random subsets of the servers to avoid the "herd effect" that can cause systems to exhibit poor behavior when clients chase the apparently least loaded server. Under Mitzenmacher's algorithm, if there are n servers, instead of sending a request to the least loaded of the n servers, a client randomly selects a subset of size k of the servers, and sends its request to the least loaded server from that subset. Note that when $k = 1$, this algorithm is equivalent to uniform random selection without load information and that when $k = n$ it is equivalent to sending each request to the apparently least loaded server. In addition to the formulating these k-subset algorithms as a solution to this problem, Mitzenmacher uses a fluid limit approach to develop analytic models for these systems for the case when $(n \to \infty)$; however, the primary results in the study come from simulating the queuing systems, and we follow a similar simulation methodology here.

Mitzenmacher concludes that the $k = 2$ version of the algorithm is a good choice in most situations. He finds that it seldom performs significantly worse and generally performs significantly better than the more aggressive algorithms (e.g., $k = n$ or even the modestly aggressive $k = 3$ algorithm) and that $k = 2$ outperforms the uniform random ($k = 1$) algorithm for a wide range of update frequencies.

We believe, however, that this approach still has drawbacks. In particular, we note that as T—the update frequency of load information—changes, the optimal value of k also changes. For example, under Mitzenmacher's periodic update model and one sample workload he examines, $k = 100$ outperforms $k = 2$ by more than 70% when $T < 0.1$, but $k = 2$ quickly becomes much better than $k = 100$ for larger values of T. Similarly, although $k = 2$ outperforms $k = 1$ when $T < 36$ for such a workload, the reverse is true for larger values of T. For example, when $T = 100$, the $k = 1$ algorithm is a factor of 2 better than the $k = 2$ variation.

We also note that under Mitzenmacher's algorithms, the resulting arrival rate at a server depends only on the server's rank in the sorted list of server loads, not on the magnitude of difference in the queue lengths between servers. Furthermore, the $k - 1$ least loaded servers receive no requests at all during a phase. More generally, if servers are ordered by load, with s_0 having lowest load and s_{n-1} the highest, a given request will be sent to server s_i if and only if (1) servers s_0 through s_{i-1} are not chosen as part of the random subset of k servers and (2) server s_i is chosen as part of that subset. Because the probability that any server s_j is chosen as part of the k-server subset is $\frac{k}{n}$, the probability that conditions (1) and (2) hold is