H-0242 (H0609-002) September 4, 2006
Computer Science

IBM Research Report

To Know or Not to Know: On the Needed Amount of Management Information

David Breitgand
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

Amir Nahir, Danny Raz
The Computer Science Department
Technion
Haifa, Israel

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

To know or not to know: on the needed amount of management information

David Breitgand
IBM Haifa Research Lab, Israel
davidbr@il.ibm.com

Amir Nahir Danny Raz
The Computer Science Department, Technion, Haifa, Israel
nahira danny@cs.technion.ac.il

August 21, 2006

Abstract

It is well accepted that a better management of scarce network resources helps in providing better service. However, it is often argued that due to the high complexity of management, a much more cost effective way to assure performance is just to acquire more resources. In this paper, we address this fundamental tradeoff in a rigorous way by showing exactly how much effort should be invested into management to gain the maximal benefit.

To focus our research, we consider management aspects of load balancing in distributed systems, in which incoming tasks arrive in a Poisson stream to a collection of n identical servers. A Supermarket Model studied by Mitzenmacher in 2001, suggests that using only a small amount of information about the local queue lengths of the servers and some simple randomization in job assignment, may lead to a very efficient load balancing algorithm. However, it was assumed that the management information is obtained at no cost.

In this work, we extend the Supermarket Model by explicitly incorporating the management costs. This Extended Supermarket Model (ESM) allows us to formally study the intuitively obvious tradeoff between the usefulness of management information and the cost of obtaining it. The main result presented in this paper is that for each service request rate, there exists an optimal number of servers that should be monitored. An interesting corollary of this finding is that knowing more about the global state of the system through detailed monitoring may not be only useless, but also harmful for the total quality of service provided by the system.

1 Introduction

The increased complexity of networking infrastructure and protocols and the desire to provide high quality services at the lowest possible cost, drive many organizations to deploy more network and system management tools in their networks. In general, the management process consists of acquiring the needed information regarding the state of the system, processing this information, and taking the needed measures to ensure that the system performs according to the pre-specified objectives.

Clearly, if the management system has access to more accurate up-to-date information, it can make better decisions and improve the system performance. However, obtaining the needed information requires both communication and computation resources that otherwise could be used to provide the actual service. It is thus important to identify just the right amount of resources that should be allocated to management tasks (such as monitoring) in order to maximize the overall system performance.

Consider for example a service that is being provided by a set of servers over the network. The goal of the service provider is to provide the best service (say, minimizing the service time) given the amount of available resources (*e.g.*, the number of servers). The provider can add a load sharing system (for example as suggested in RFC 2391 [9]) and improve the service time. However, the same resources (budget) can be used to add additional servers to the system and thus provide better service to end customers. The dilemma here is between adding more computational power and adding management abilities, where the goal is to achieve the best improvement in the overall system performance. Note that in order to be effective, the load sharing system needs updated load information from the servers. Handling such load information requests requires small but nonzero resources (*e.g.*, CPU) from each server. Thus, it is not easy to predict the actual amount of improvement expected from preferring a specific configuration.

The tradeoff between committing more resources to monitoring in order to improve the quality of management decisions on the one hand, and degrading the total quality of service due to excessive monitoring on the other hand, is well known to the practicing network administrators. Presently, administrators deal with this tradeoff based on their empirical experience. Somewhat surprisingly, a formal study of this problem is noticeably absent from the literature, and to the best of our knowledge, this paper is the first step in this direction.

In order to be able to provide a quantitative study of this fundamental problem, we have to narrow down the scope of the discussion and to focus

on one specific problem. To this end, we consider management aspects of load balancing in distributed systems, in which incoming service requests arrive as a Poisson stream to a collection of n servers.

A model describing such a system was studied in [8] by Mitzenmacher. This model, termed the *supermarket model* uses a very simple randomization strategy: when a new task arrives, $d < n$ servers are selected uniformly at random, and the task is assigned to the server with the shortest queue among these d chosen servers. For $d = 1$ this process simply assigns jobs to servers uniformly at random, regardless of their load. However, for $d = 2$, the job is assigned to the least loaded server (the one with the shortest queue) among the two randomly chosen servers. It is shown in [8] that this simple process results in an exponential improvement of the expected overall time in the system compared to the random assignment scheme. Further increasing d improves the expected service time linearly. The results of [8] suggest that even a very small amount of management information coupled with random job assignment may lead to a very efficient load balancing strategy, and as we use more information we keep on improving the scheme. However, this study assumes that the management information is obtained and processed at no cost. As explained above, in many scenarios this assumption is not realistic.

We extend the supermarket model by incorporating the management costs into it. In particular, we assume that when a server is polled about its load, it has to allocate resources in order to answer this query. In other words, when a server answers a monitoring inquiry, it can spend less CPU cycles on processing the actual service requests. An important factor is the ratio between the time it takes a server to answer a load request and the mean expected service time of a job. This *load monitoring efficiency ratio*, denoted by C, reflects the amount of disturbance the monitoring task has on the actual service. The overall capacity reduction of a server is also proportional to the number of monitoring inquires per time unit, which depends linearly on d. When d increases we have more information and thus the expected average time in the system decreases thanks to better load sharing. On the other hand, each server becomes more affected by load queries and thus the service time (and therefore also the overall time in the system) increases. The *Extended Supermarket Model (ESM)* allows us to rigorously study this intuitively obvious tradeoff between the usefulness of of management information and the cost of its maintenance.

The main result presented in this work is that for each system load and monitoring efficiency ratio C, there exists an optimal number d^* of servers that should be monitored in order to minimize the overall expected time of

jobs in the system. One of the corollaries of this finding is that knowing more about the global state of the system through detailed monitoring may be not only useless, but also harmful for the total quality of service.

The practical implementation of these theoretical results depends on the actual deployed load sharing scheme. Very common architecture is a centralized load balancing device that assigns the incoming service requests to the different servers. The assignments of the jobs to the different servers can be done using Load Sharing NAT (LSNAT) as described in RFC 2391 [9] or via TCP Splicing [1]. The load sharing algorithm running in this device decides on the actual server that should handle each job, and any adaptive load sharing algorithm must have feedback regarding the load at each server. The main problems with this architecture are its lack of scalability, and the existence of a single point of failure. To address these shortcomings, one can use a distributed scheme in which the client is responsible to measure the load on several servers and to choose the least loaded one. From the servers' point of view, this scheme is not much different from the centralized one, since in both cases d load requests are sent out per each incoming service request. In both cases, the load sharing decision time is not consider in the overall time the job is in the system. We refer to these schemes as the *centralized* and *distributed* ESM respectively.

It should be noticed, however, that the centralized load sharing device can poll the servers load periodically and not per request. This reduces the number of load inquiries but depending on the polling rate may affect the load balancing quality due to the staleness of the data. We address this issue in Section 6.1 of this paper. In Section 6 we also address a different variant of the model in which the servers themselves participate in load sharing decision taking. In this variant of the model, called Server Based ESM (SBESM), clients pick up servers uniformly at random. When a job arrives to a specific server, it inquires the load on $d-1$ other randomly chosen servers, and assigns the job (*i.e.*, forwards the service request) to the one with the shorter queue (including itself). In this case the load sharing work is done by the servers and thus further reduces their ability to serve actual clients. Yet, this scheme is very scalable, it does not assume that the clients can perform load inquires, and it is very likely to be used in future distributed service environments like Web Services [4].

Our main result - the effective existence of an optimal number of servers d^* that should be checked - apply to all versions of the model, and thus to the entire spectrum of load sharing schemes. We support our theoretical findings by conducting an extensive simulation study indicating that the theoretical models predict the actual behavior well. We also implemented

the Server Based ESM on a set of distributed servers and measured the actual performance of such a system. Our results indicate that the theoretical results can be used to improve the system performance considerably.

In today's complex systems, management costs are inherent. Even though they can be reduced by configuring a system in a different way, or by using a different algorithm, they cannot be avoided completely. This paper presents a novel approach that can allow rigorous reasoning about the theoretically interesting and practically important tradeoff between management costs and the overall system performance.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 presents ESM. Section 4 evaluates ESM through simulation. In Section 5 we discuss the optimal parameters of ESM. Section 6 discusses practical implementation issues and evaluates ESM through emulation on a real computer system. Section 6.1 discusses the centralized load sharing scheme that uses periodic polls. Section 7 concludes and provides some future directions.

2 Related Work

Load balancing is concerned with distributing processing and traffic as evenly as possible across a computer network so that no single device is overwhelmed and the total quality of service increases. Busy Web sites typically employ multiple Web servers in a load balancing scheme. If one server becomes overutilized, requests are forwarded to another server(s) with more spare capacity. Similar approach is being taken in parallel computing, computational grids, *etc.*. Since communication resources are finite, load balancing of communication channels is also required in practical settings.

There are two basic forms for load balancing: static and dynamic. For static load balancing (*e.g.*, DNS based load balancing), monitoring of a system state is not required. Requests are distributed across the network using for example Round Robin policy or by randomly selecting a server and/or a communication channel. As a consequence, static load balancing schemes are not adaptable and often sub-optimal. See [6] for a formal study of the adaptable Least Loaded Routing policy that forwards requests to the least loaded server. As [6] shows, an adaptable load sharing policy that takes the current state of the system into account is provably superior to a static non-adaptable one. In this work we concentrate on adaptable load balancing algorithms.

In their milestone paper [3], Azar *et. al.* introduced the supermarket

model (see previous section) for evenly spreading a finite number of items among n identical locations. Azar *et. al.* showed that by simply making two random choices ($d = 2$) and selecting a location with the least number of items already assigned (ties is broken arbitrarily), one reduces the maximal number of items per single location exponentially. Further increasing d results in linear decreasing of the maximal number of items per location. Azar *et. al.* studied a closed finite system in which items never leave the system and the load balancing process terminates when all items are assigned to their locations.

An application of the supermarket model to dynamic load balancing, where an infinite number of service requests arrive from a stream with a given traffic intensity and where clients leave the system once their request is serviced, was done by Mitzenmacher [8]. In this work, an infinite stochastic supermarket model is limited by a set of deterministic differential equations describing the queue length dynamics. It turns out that results, similar to [3] hold, but the analysis is much more complicated.

As was already noticed in the introduction, the supermarket model of [8] does not take the cost of acquiring the local state of the servers into account. In [7], a question of how often the local state of the servers should be polled, is addressed. It turned out that obtaining a closed form solution for the expected average time in the system is difficult in this case. Therefore, [7] resorted to an extensive simulation study, showing that randomness is a powerful tool to curb *herding effect*, which becomes a dominant factor in performance degradation as the system state is acquired periodically with decreasing rate. The effect of periodic updates on the ESM performance is studied in this paper in Section 6.1.

In [2] a system somewhat similar to Server Based ESM (see Section 6), is studied. The primary goal of that work is to achieve an autonomic adaptable strategy for peer-to-peer overlay maintenance with QoS guarantees in spite of possible server crashes. The Authors use an epidemic algorithm to construct an overlay of connectivity d with randomized membership (*i.e.*, periodically the neighborhood of each server is reshuffled randomly). The servers in the neighborhood share their load information periodically. A service request from an external client is assigned randomly to some server and gets forwarded to a less loaded peer in the neighborhood of this server. When a server's load gets beyond some threshold, the server is considered overloaded. If all neighbors in the neighborhood of this server are overloaded, a server may pick random servers outside of the neighborhood and ask them to join in and share the load. It was demonstrated trough simulations that by controlling the parameter d efficient QoS management strategy

is attainable. Similarly to [6, 8, 7] and this paper, [2] considers average time in the system as performance parameters that should be minimized. However, even though the monitoring process plays a pivotal role in [2], its cost is not explicitly accounted in the simulations.

To the best of our knowledge, the current state of the art in performance modelling is that the management costs, and in particular the monitoring costs are neglected. This creates a gap between the theoretical models and the practical implementations, in which these costs are important. Our focus in this work is the formulation, analysis and evaluation of a performance model for adaptable load balancing where such a model explicitly accounts for the inherent management expenses.

3 The Extended Supermarket Model (ESM)

Following the supermarket model, we consider a system that consists of n identical servers. Each server processes its incoming service requests according to the FIFO discipline. Jobs arrive to the system in a Poisson stream of rate $\lambda \cdot n$, $0 < \lambda < 1$, service time is exponentially distributed with mean 1. In the distributed ESM model depicted in Figure 1 (b), when a customer submits a request, it selects $d < n$ servers uniformly at random (with replacement) and sends d inquiries about the length of the server queue to each of the selected servers. These monitoring requests have a precedence over the actual service requests, *i.e.*, upon receiving a monitoring request, the server preempts the currently running job (if any exists) and answers the load request immediately. We assume that processing the monitoring request takes a fraction $0 < C < 1$ of the mean service time of the actual service. This factor is the load monitoring efficiency ratio that reflects the fraction of the (CPU) resources needed in order to answer a load request. When the client obtains all d answers (we assume that there are no message losses), it selects the server with the minimal queue length (ties are broken arbitrarily) and forwards the job to this server.

In the centralized ESM model depicted in Figure 1 (a), all the requests arrive at the centralized load balancing device that queries d random servers upon the request arrival (periodically load update is dealt with in Section 6.1) and forwards the job request to the server with the minimal queue length.

Note that we do not model the time it takes the load inquiries to get back to the client or to the centralized load sharing device, or the processing time at the decision at the client side or at the load balancing device. The

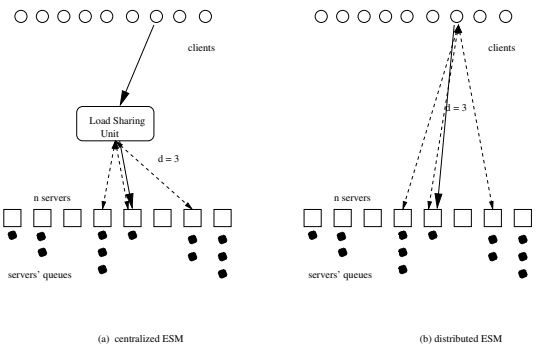

Figure 1: The Extended Supermarket Model

actual time in the system starts when the job arrives to the selected server.

The mean service time and thus also the mean service rate is 1. However, since the servers answer load inquiries immediately, the *effective* service rate is smaller than 1. Job requests arrive at rate $\lambda \cdot n$ (where the time unit is such that the service time is one) and each job creates d load inquiries. Since we have n servers and they are chosen uniformly at random, each server gets a Poisson load request stream with incoming rate of $d \cdot \lambda$. Thus, handling each load requests takes C time units and the effective service rate is:

$$\mu' = 1 - \lambda \cdot d \cdot C \quad . \tag{1}$$

Now we define $\rho = \lambda/\mu'$, to be the arrival rate normalized by the effective service rate, and we get:

$$\rho = \frac{\lambda}{1 - \lambda \cdot d \cdot C} \quad . \tag{2}$$

Clearly, ρ must be smaller than one in order to keep the system stable. In other words, if we want to have a stable state where the queues in the system have finite length we must have $\rho < 1$, or

$$\lambda < \frac{1}{1 + \lambda \cdot d \cdot C} \quad . \tag{3}$$

Note that Equation 3 indicates that in some cases where the load is high, the monitoring process can push the system into an unstable state. This means that in some cases we would be better off without any management whatsoever.

We follow the footsteps of [8] and define $n_i(t)$ to be the number of servers with exactly i jobs in their queue (This includes the job that is being served). Next we define

$$s_i(t) = \sum_{k=i}^{\infty} \frac{n_k(t)}{n} \tag{4}$$

to be the fraction of the servers with at least i jobs in the queue. When not needed we omit t; clearly $s_0 = 1$ and s_1 is the fraction of non empty servers.

It is easy to see that:

$$\sum_{i=1}^{\infty} s_i(t) = \frac{1}{n} \sum_{i=1}^{\infty} i \cdot n_i(t) \tag{5}$$

is the average queue length at time t.

For any $d > 1$, C, finite n, and λ that satisfy Equation 3, the system is in stable state, and when t is large enough there is a fixed probability to be in each of the states defined by the vector $(s_0, s_1, s_2, \ldots)$.

In such a case, a new job joins a server with a queue of size i only if all d chosen servers have queues not smaller than i, and at least one of them has a queue of size i. This happens with probability $s_i^d - s_{i+1}^d$. Similarly, the probability that a job is finished from a server with queue of size i is $s_i - s_{i+1}$. This implies that the following differential equation holds for $i \geq 1$:

$$\frac{ds_i}{dt} = \rho(s_{i-1}^d - s_i^d) - (s_i - s_{i+1}), \tag{6}$$

where $s_0 = 1$. This set of equations has a unique fix point (see (1) in [8]).

$$s_i = \rho^{\frac{d^i - 1}{d-1}}. \tag{7}$$

Now, in order to compute the expected time a job spends in the system, we could use the method described in Section 2.4 of [8], or use Little's Theorem [5], and divide the expected queue length by λ. Note that we need to divide by λ and not by ρ since we want to have the expected time in the system in units of the service rate and not of the effective service rate that depends on C and d. We get:

$$T_d(\lambda) = \frac{1}{\lambda} \sum_{i=1}^{\infty} \rho^{\frac{d^i - 1}{d-1}} = \frac{1}{1 - \lambda \cdot d \cdot C} \sum_{i=1}^{\infty} \rho^{\frac{d^i - 1}{d-1} - 1} =$$

$$T_d(\lambda) = \frac{1}{1 - \lambda \cdot d \cdot C} \sum_{i=1}^{\infty} \rho^{\frac{d^i - d}{d-1}} \tag{8}$$

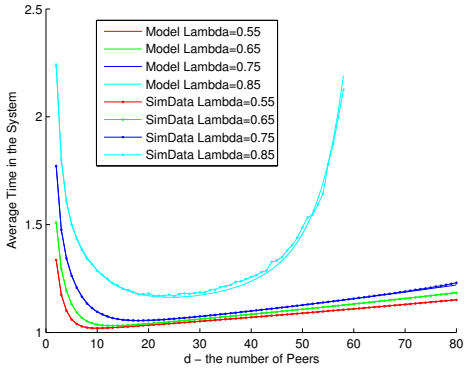

Figure 2: Model Vs. simulation low load

4 Simulations

Before we proceed to the simulation analysis of $T_d(\lambda)$, we would like to develop some intuition about the function behavior. As shown in [8], in the original supermarket model, $T_d(\lambda)$ is a monotonically decreasing function for any $0 < \lambda < 1$. However, Equation 8 suggests that as d increases, the denominator decreases, resulting in higher waiting times, and when $\lambda \cdot d \cdot C$ approaches 1, the waiting time in the system goes to infinity. Thus, we expect $T_d(\lambda)$ to decrease when d increase, but at some point as d keeps on increasing, the value of $T_d(\lambda)$ should increase as the servers put more and more resources into monitoring the load and this affects the service time.

In order to verify that the Extended Supermarket Model (ESM) indeed models correctly the behavior of sever system as described in the previous sections, we conducted an extensive set of simulation runs. This is done using an in-house event driven simulation software. The Centralized ESM is simulated for different load, d, and C values.

Figure 2 is obtained through simulating the Centralized ESM with 300 servers for $C = 0.003$. We plotted the simulation results and the model prediction obtained from Equation 8 on the same graph. Each simulation sequence contained 1,000,000 jobs, and each value in the graph is the average of 10 such runs, where the values of standard deviation were well below 1% of the obtained service times for all points except for $\lambda = 0.85$ and $d > 40$ where the standard deviation values were up to 20%.

One can see that the expected behavior indeed realized. The expected

time in the system decreases when d increases, and at some point it starts increasing. When the system load is low (*e.g.*, $\lambda = 0.55$) the value of $T_d(\lambda)$ drops from 1.34 for $d = 2$ to about 1.02 for $d = 10$, and then it increase almost linearly as d increases. This is due to the fact that when d increase from 2 to 10 the load sharing quality increases, while the effect of the cost devoted to monitoring is still small, however at this point, the return from increasing the number of peers becomes negligible while the effect of the monitoring cost keeps on increasing linearly.

When the load is higher (e.g., 85%) one can see that there is a point in which $T_d(\lambda)$ starts increasing very fast. This is happening when the system approaches the instability point predicted by Equation 3. In this area the results becomes much more noisier, and the standard deviation of the simulation results increase.

The parameters chosen here reflect normal working conditions of distributed servers systems. A cluster size of 300 servers may seem to be large, but this is not entirely far-fetched. Even today there exist many organizations that deploy 300 and more server farms (*e.g.*, Yahoo, Google, *etc.*). The value of the load monitoring efficiency ratio (C) was chosen to be 0.003 according to the data from the system deployment over our tested as described in Section 6. One can observe that the model predicts the system behavior very accurately in this range even when we used $n = 300$ servers in the simulation and the model deals with $n \to \infty$. The effect of the umber of servers n is very small all we need is a large enough ($n > 100$) number. This is due to the fact that the system load is $n \cdot \lambda$ and therefore the average load on each server depends only on the load parameter. In fact we ran several simulations for different values of n and similarly to the results of [8] the behavior is the same but the accuracy of the simulation improves as n increases.

Figure 3 depicted the simulated results and the model prediction for high load values. Again $C = 0.003$ and each point is the average of 10 runs each produced from a set of 1000000 jobs. The number of peers in this case goes from 2 to 40, and the number of servers was 500. One can observe similar behavior, and the precision of the model is quite well, except when ρ approaches one, and the system becomes less stable.

5 Optimal Load Balancing in ESM

In order to get the best d for a fixed set of parameters one needs to get the derivative of $T_d(\lambda)$ with respect to d. This is a rather complex expression

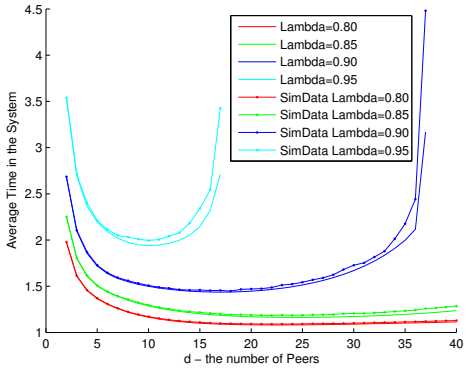

Figure 3: Model Vs. simulation high load

and it has no closed formulation. Figure 4 depicts the optimal number of peers (d) as a function of the system load for different load monitoring efficiency ratio (C) values. One can see that for relatively high values of C (i.e., $C = 0.02$ where a load query takes 2% of a job service time) the optimal number of peers increases slightly as the load increases over 50% but it does not go over 10, and at around $\lambda = 0.7$ it start decreasing. This is happening since the the monitoring cost is relatively high, and cannot be justified by the improvement in performance due to better load balancing. Furthermore, when load is high, the number of peers must be small in order to keep the system stable (as indicated by Equation 3). When the load monitoring efficiency ratio values decreases (i.e., $C = 0.001$ where a load query takes only 0.1% of a job service time) the picture is quite different. In this case the optimal d increases significantly as the load increases and reaches a value of 46 at $\lambda = 0.9$. This is due to the fact that the monitoring cost is relatively low and it is worth to improve the average time in the system by improving the load sharing quality. When the load increase over 90%, the cost effect becomes dominant and thus the optimal number of peers decreases sharply.

Figure 5 depicts the optimal number of peers (d) as a function of the load monitoring efficiency ratio (C) for different load values. As one can observe, the behavior depends strongly on the value of the load. For low load values (e.g., 50%), the optimal number of peers is around 10, and it decreases very slowly when C increase. This is due to the fact that when the load is low, even a small number of peers results in almost optimal load sharing since

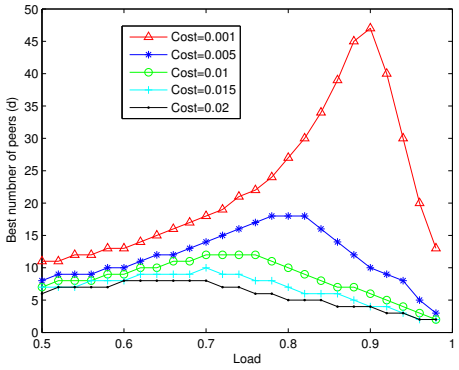

Figure 4: Best number of peers as a function of the load

many of the servers are in fact idle, and increasing d only increases the cost without any significant performance improvement. For higher values of λ (e.g., 90%) the value of C has a much stronger effect on the optimal number of peers. This is happening since a small change in C has a significant effect on $1 - d \cdot \lambda \cdot C$. When the load is even higher (e.g., 98%) even when C is small, a small number of peers pushes the system toward instability, thus the optimum numbers of peers, even in that region, is small.

6 Practical Implementation and Evaluation

In this section we deal with practical implementations and actual evaluation of the theoretical results developed in the previous sections. We start by evaluating the effectiveness of periodic load update model (as opposed to per request load inquiry) in the centralized ESM. Next we describe the Server Based ESM (DSBESM) and compare it to the centralized and distributed versions of the model. Then we evaluate the actual performance of a load sharing system we implemented using this model and examine the actual benefit one can gain from the rigorous study of monitoring cost.

6.1 Periodical Update Model – A Simulation Study

In practical implementations, the monitoring information (in our case the load of the servers) is often acquired using periodic updates. This is done in order to conserve communication and processing resources of the network.

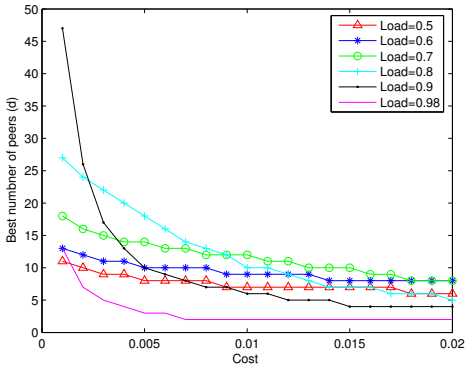

Figure 5: Best number of peers as a function of the cost

One of the more important generic management problems is determining an optimal frequency at which the management information should be gathered.

The trade-off here is intuitively clear. If we poll the servers load often, then the monitoring cost is high, but the load information is up to date, and therefore the load balancing decisions are better. On the other hand, the less frequent are the updates, the lower is the cost of monitoring, but then the local information becomes stale, and the overall quality of the load balancing process decreases.

In [7], a periodic updates model called the Bulletin Board model was studied. The goal there was to estimate the penalty to the load balancing process due to the use of stale information. The motivation for this work is the drive to reduce the management overhead, but the monitoring cost was neglected and is not part of the Bulletin Board model. It turned out that obtaining a closed form solution for the expected time in the system is very difficult in this case, even when one does not consider the cost of obtaining the load information. Consequently, [7] used simulation analysis.

In this section, we use a similar simulation study to investigate the effect of periodic updates in ESM, when monitoring cost is part of the model. We consider the centralized ESM, but the load sharing device uses periodic load polling instead of the per job load polling mechanism. Every q units of time, all servers are polled for their load (i.e., their queue size), this information is kept in a local data base at the load sharing device. When a new job arrives, the load sharing device chooses d servers uniformly at random, and forwards the job the server that has the smallest queue according the information in

the local data base. As in the centralize ESM, the load answer by each server during the load poll takes a fraction $0 < C < 1$ of the mean service time of the actual service request. However, unlike the centralized ESM case, using a larger value for d does not create more load on the servers since the load is a function of the update rate $\frac{1}{q}$, and d only affects the number of lookups in the local data base. It is important to note at this point that the time the job is waiting for the server assignment in the centralized load sharing device is not part of the total time in the system as defined for the ESM model in Section 3. In order to be able to compare findings of this sections with the results we presented so far, we keep it this way here, and we address this issue when we described the Server Based ESM in the next section.

We simulated a cluster of 300 servers, processing 100000 jobs that arrive from a Poisson streams with different traffic intensity (*i.e.*, load). Each simulation run is repeated 10 times and the average of these runs were computed to produce the results. Similarly to Section 4, we used the load monitoring efficiency ration $C = 0.003$.

Figure 6 shows dependency of the average time in the system on d (number of monitored peers) for traffic intensity $\lambda = 0.55$. Different curves on the graph correspond to different load update periods. The per-job load polling policy serves as a baseline. One may notice that for the low traffic intensity case, periodic updates do not yield substantial savings even for large values of d. Moreover, the actual mean times in the system obtained using periodic polling are marginally higher than the baseline for smaller d-s. One can also observe that in the tested parameters there is no benefit to use d values that are larger than 20 since increasing d only gives marginal improvement (if any) in the average time in the system.

There are two primary reasons for this behavior.

- When $\rho = 0.55$, the queues are empty most of the time. Therefore, on the one hand, using CPU cycles for updating the load bulletin board delays only a small fraction of jobs. On the other hand, the information gain from the load updates is low. In fact, most of the time these updates show zero length queues at many servers.

- Since the periodic updating of the load bulletin board is done synchronously at *all* servers, chances that a jobs, which considerably deviate from the mean service time would get preempted and delayed are marginally higher than those in the case when *only a fraction* of servers is polled for their queue length upon a new job arrival.

Similar behavior is observed when the system load is increased to be

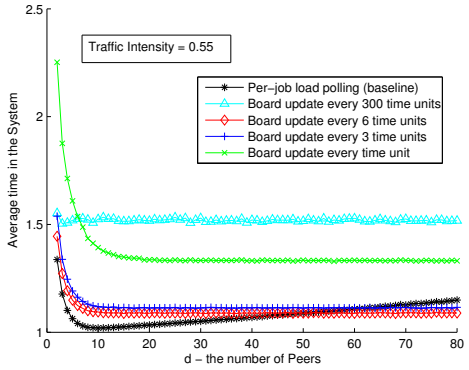

Figure 6: Average Time in the System for ESM with Periodic Updates

$\lambda = 0.75$, but the figure is omitted due to the lack of space.

Figure 7 shows the time in the system as a function of the centralized load polling period length for $\rho = 0.55$. As one can see, when the update period increases beyond a certain value, the total quality of the load balancing decreases due to staleness of the data. We observe the same trend for all traffic intensities. The pace of the service degradation to the management data staleness is different in each case, though. Not surprisingly, the higher is the traffic intensity, the faster is the load balancing degradation due to longer update periods.

One may think that when the system becomes more heavily loaded, the advantages of the periodic updates would become evident. Figure 8 shows the system behavior for $\rho = 0.85$. Indeed for a large fixed value of d (60), using the appropriate update rate (every 10 time units) periodic updates shows almost a factor of 2 improvement compared to the per-job load inquiry case. However, for smaller d values or higher rate of load update, this is no longer true.

Figure 9 shows the time in the system as a function of the centralized load polling period length for $\lambda = 0.99$. Note the similarity to Figure 7 in the curve's shape. One important corollary of this simulation study is that similarly to existence of d^*, which is the optimal number of polled peers, there exist an optimal rate of polling for each fixed d. To gain minimal time in the system, the administrator or an automated policy may vary either of these two parameters.

If we compare per-job polling with optimal d values as described in

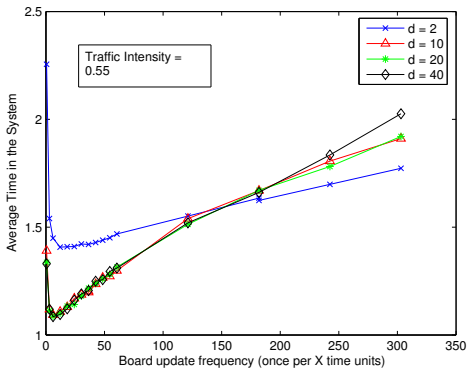

Figure 7: Average Time in the System as a Function of Update Rate ($\rho = 0.55$)

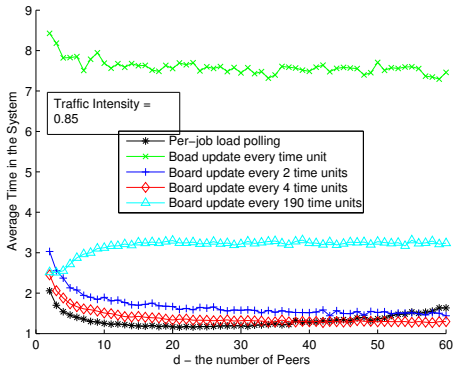

Figure 8: Average Time in the System for ESM with Periodic Updates ($\rho = 0.85$)

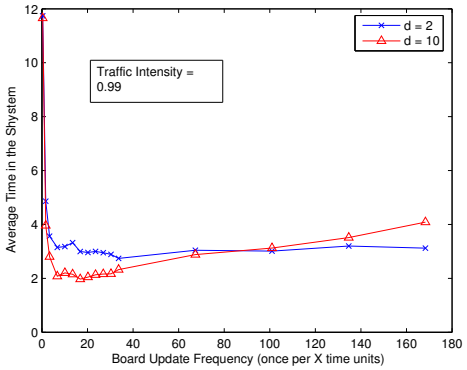

Figure 9: Average Time in the System as a Function of Update Rate ($\rho = 0.99$)

Section 5, to the optimal rate we find here, we still see that optimally tuned per-job load inquiries result in better mean times in the system. Indeed, periodic updates can be deployed without hurting the total quality of load balancing, but this does not improve the total system cost effectiveness. This may sound somewhat surprising, since this counters a popular belief that periodic updates in some form or another is always more cost efficient than the per task polling.

6.2 The Server Based Extended Supermarket Model (SBESM)

Both in the centralized and distributed versions of the ESM (see Figure 1) the load inquiries are sent to the servers and they affect the system's behavior, but the actual load sharing process is done outside the system. The resources (communication and computation) used for making load decisions are not part of the system's resources. In the centralized ESM model, we assume an additional load balancing device, and in the distributed ESM model, we assume that the clients themselves perform the load sharing process using their resources. In addition, the time elapses from the time the service is requested until the time a server is specified was not considered as part of the total time in the system. Indeed, this additional time could be considered to be a constant additive term since load inquiries have total priority, and since all d inquiries can be done in parallel, but one must note that it is not part of the model discusses so far.

As mentioned before, the centralized ESM has two major problems. The

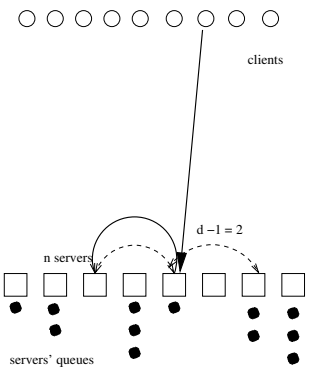

Figure 10: The Server Based Extended Supermarket Model (SBESM)

first one is its lack of scalability, and the second one is the existence of a single point of failure. One way to address these shortcomings, is to use the distributed scheme in which the client is responsible to measure the load on several servers and to choose the least loaded one. However, this option also has several problems. To start with, clients need to have access to the list of servers so that they could choose d servers and inquire them regarding their load. Secondly, there should be a protocol that is standardized and used both by clients and by the servers. This is definitely a problem that cannot be solved in the forthcoming future.

There is a need then, to have the load sharing responsibility with the system, yet not to do it in a centralized load sharing point. This can be done in the following way. Clients sent their job requests to a randomly chosen server, using the DNS mechanism for example. When a job arrives to a specific server, it inquires the load on $d-1$ other randomly chosen servers, checks the local load, and assigns the job to the one with the shorter queue (including itself). If the job is to be assigned to a different server the request is forwarded, and if it is assigned to the local server (since the local queue is the shortest one) the request is moved to the end of the local queue. Figure 10 depicts this scheme.

Note that in this case the load sharing work is done by the servers themselves and thus further reduces their ability to serve actual clients. Our model should consider this additional requirement from the servers, and the service rate should be reduced accordingly. Service requests arrive at rate of $n \cdot \lambda$. Since servers are assigned uniformly at random, each server sees a Poisson stream of rate λ. It needs to handle each job in this stream by

creating $d-1$ load inquiries, sending them to randomly chosen peer servers, getting the actual loads from the servers and choosing the one (including the server itself) with the shortest queue. Clearly the work is proportional to $d-1$, but it is not clear that we actually should use the same constant C - the load monitoring efficiency ratio, since creating and handling the requests may require different resources than answering a load query. Using an additional parameter α reflecting this point, we get that the effective service rate in this case is:

$$\mu' = 1 - (1+\alpha) \cdot \lambda \cdot d \cdot C \quad . \tag{9}$$

and the expected time in the system is:

$$T_d(\lambda) = C + CT + \frac{1}{1-(1+\alpha)\cdot\lambda\cdot d\cdot C} \sum_{i=1}^{\infty} \rho'^{\frac{d^i-d}{d-1}} \ , \tag{10}$$

where ρ' is the normalized rate in this case and CT is the uppaer bound on the Round Trip Time (RTT) between servers. The RTT is given in mean service time units and it reflects the fact that all $d-1$ queries are done in parallel, and they are answered by each server within C time units. In the next section we use this formalism to analyze the observed performance of the load sharing system we implemented.

6.3 The implementation of server based load sharing system

In order to study the practical performance of a distributed load sharing system as described in this paper and the actual usefulness of the theoretical results, we implemented a Server Based ESM system and tested its performance on a tested network. To save space we omit the implementation details and only highlight the more important points.

We deployed 11 copies of SBESM servers on heterogeneous hosts over a local area network. The machines used were all Pentium 4, with 1.7-3GHz and 0.5-1GB of RAM, 3 of them running Windows XP and 8 Linux. A client was created to generate all requests, to collect all answers from the servers and to produce logs that were processed to create the system statistics. The average service time in all test runs was set up to be around 0.33 second. Client used UDP transport protocol to communicate with the servers. Due to the versatility of the machines there was a factor of up to 1.5 between the service time in the fastest machine and the one in the slowest machine. From testing the service time for different values of peers (d) we found that on the average the service time increases by 2ms when d increase by 1 (from

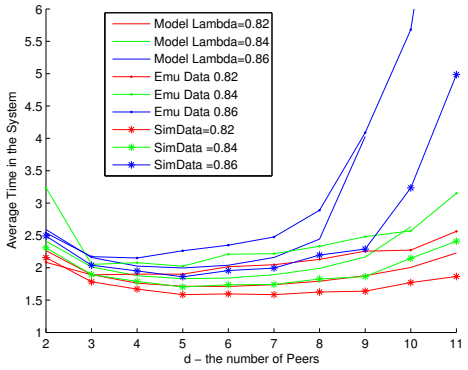

Figure 11: Model Vs. simulation low load

2 to 11). Taking α in Equation 10 to be 1 we get that C is $\frac{1}{330}$ or about 0.003, which is the value used throughout this paper.

For each d value and each load parameter (load is achieved by tuning the average inter arrival time at the client) we performed 9 runs of about 12000 job requests each (jobs were simulated as CPU intensive busy loops), and computed the average time in the system. Due to the usage of UDP some of the requests or the reports containing the job statistics were lost, but the number of lost jobs was less than 1% of the jobs in all test runs. The average time in the system for three load values as a function of d is depicted in Figure 11. One can see that both the model and the simulation predict the actual behavior quite well. It is also notable that even for a relatively small number of servers (11 in our case) and even when the cost contains also the load sharing cost (*i.e.*, the monitoring and decision costs), the optimization gain from using the scheme can get up to 15-20%. That is, by tuning the system (i.e. choosing the optimal number of peers) one can get a 20% reduction on the overall time in the system. As shown in Section 4 this gain will be larger for a larger number of servers.

7 Conclusions and Future Work

Many organizations are facing the following dilemma: what is more cost effective, investing in management or in actual service. In the context of networking and network management this translates into deciding how much resources should be put into network management. On one hand, better

management may allow the network to be more efficient and hopefully to provide better QoS to end customers, but on the other hand, the actual benefit from management tools is not always obvious, and the same resources could be more useful when put directly into service infrastructure (buying more bandwidth for example).

This paper addresses this point by developing a formal model that captures both the cost of and the benefit from management processes, concentrating on load sharing among distributed servers. We define the Extended Supermarket Model, analyze the cost effectiveness of monitoring load in such an environment, and through extensive simulations and performance evaluation of a testbed implementation demonstrate its usefulness in practical settings.

As in many cases, there is a trade off between the complexity of the model and its ability to capture the core parameters that have the most significant impact on the system's behavior. In this respect, it appears that the centralized and distributed ESM captures the essence of the management/service tradeoff in the appropriate abstraction level. The only parameter used, except the system load, is the load monitoring efficiency ratio, C, yet the model describes the main aspects of the system behavior very accurately. Trying to capture more detailed aspects of real practical systems results in the need to use more parameters (such the update rate in the periodic update model, or the α and the RTT in the server based version), and thus in adding more complexity and loss the clarity of the presentation. This paper is just the first step in the formal study of this management dilemma. One should use similar techniques in order to understand additional aspects of this tradeoff in the networking field at large.

References

[1] A. Cohen and S. Rangarajan and H. Slye . On the performance of TCP splicing for URL-aware redirection. In *2nd USENIX Symposium on Internet Technologies and System*, Boulder, CO, USA, October 1999.

[2] C. Adam and R. Stadler. Adaptable Server Clusters with QoS Objectives. In *9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, Nice, France, May 2005.

[3] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 2000.

[4] IBM Developer Network. SOA and Web Services.

[5] J. D. C. LITTLE. A Proof for the Queueing Formula L = kW. *Operations Research*, 9(3):383–387, 1961.

[6] M. Alanyali and B. Hajek. Analysis of Simple Algorithms for Dynamic Load Balancing. *Mathematics of Operations Research*, 22(4):840–871, 1997.

[7] M. Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel and Distributed Systems*, 11(1):6 – 20, January 2000.

[8] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094 – 1104, October 2001.

[9] P. Srisuresh and D. Gan. Load Sharing using IP Network Address Translation (LSNAT), RFC 2391 1998.