RC 21933 (98366) 18 October 2000 **Computer Science**

IBM Research Report

A Load Balancing and Resource Management Strategy for a Distributed Database System

Norman Bobroff, K. A. Beaty, G. Bozman
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

A Load Balancing and Resource Management Strategy for a Distributed Database System

Norman Bobroff, K.A. Beaty, and G. Bozman
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

Abstract:

We offer a solution to the problem of load balancing a system of heterogeneous database applications within a group of host servers. The problem is studied in architectures where the load has a strong affinity to a database, file, or application that can be selectively placed on any of the servers. There are three important timeframes in which load leveling is addressed. 1) Long-term usage patterns that are sometimes predictable from historical data. 2) Variations from minutes to hours caused by clients initiating and breaking persistent sessions. 3) Short term fluctuations arising from basic queuing behavior caused by a large set of independent clients generating data requests or remote procedure calls against the applications. A key point is that load leveling should be done both to optimize long-term average processor utilization and to minimize load variations that cause the service time to become unstable or unpredictable due to the formation of large service queues. We demonstrate the importance of balancing both total load and workload profile of applications statically placed on each server. The coefficient of variation is shown to be minimal when applications are placed to balance resource profiles. Balancing profiles based on historical usage is also shown to equalize long-term resource growth rates amongst the servers. We study techniques of compensating mid-timescale variability by direction of persistent client connections to least loaded server. A comparison of client redirection methods shows that an efficient leveling scheme is client redirection to the best of two randomly selected database replicas. Load balancing on short timescales is typically addressed by dynamic routing of requests in a server cluster. It is often desirable to simultaneously apportion resources other than load such as network or disk utilization. In fact, we argue that a multidimensional approach to resource management can provide a more stabile balance for each resource than one-dimensional approaches. This study is performed largely within the context of Lotus Notes and similar two tier application architectures. The conclusions are based on analysis, simulation, and data from Lotus Notes database servers at IBM.

1. Introduction

This paper describes load balancing and resource management solutions for client-server architectures in which application code primarily accesses files and databases on the same server. In this architecture, client access to an application results in load primarily on the server hosting the application. As a consequence, a server's load is largely determined by the applications it hosts. A key extension of the architecture is the selective creation of copies or replicas of heavily accessed application databases and files. Replicas are placed on different servers, and a mechanism is provided to maintain data coherence across the replicas. Figure 1.1 shows this structure in which M servers host applications to N clients. Application B is replicated across two servers, while the other applications have a single instance. The degree of data consistency and frequency of updates between replicas is based on the application requirements. The creation of replicas means that server load is determined both by the single instance applications on a server, as well as client affinity to the replicated applications on that server. Establishing data coherence between replicas imposes additional server load when there are frequent updates. While this cost can be an important consideration in load balancing, replication provides substantial flexibility in resource management. Replication is also a means of increasing application availability as well as facilitating updates and consistency of application versions. The advantages and costs of replication are considered in some detail in section 4 of this paper.

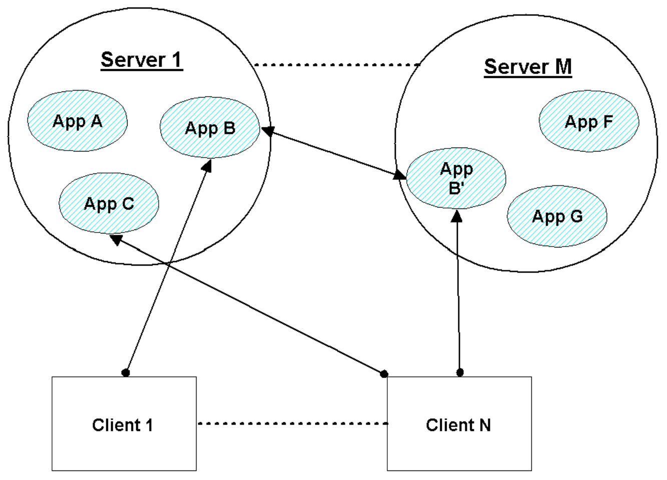

Figure 1.1. The architecture in which migration and replica creation and client redirection play a role in managing load and other resources.

It is useful to categorize database applications according to whether they are accessed primarily by a single client, or by multiple clients. A ubiquitous example of the former is an individual's electronic mail database. In this paper, a single user database is often referred to as a mail database, while a multi-user database is termed an application database. In general, the only reason to create server replicas of single user applications is for application availability. The Lotus Notes database system (described in more detail below) provides for client side databases that can be accessed locally. In this instance, local replicas clearly reduce server load by offloading read activity. This is an effective mechanism of reducing load, particularly for mail servers.

The application architecture of figure 1.1 supports several mechanisms to manage server load. One is the decision of where to statically place applications and databases. A second is the selective creation and placement of replicas of heavily used applications. Another is the ability to dynamically route clients to database replicas on lightly loaded servers. A further degree of freedom is the potential to dynamically create new replicas in response to load. This paper emphasizes the first three of these strategies. Server load and application service times in the architecture of figure 1.1 can vary from short-term surges to long-term changes in average utilization caused by evolving client behavior, new application deployment, and growing system capacity. Thus, it is useful to partition the problem of load balancing in the time domain. This approach is outlined in figure 1.2. Here we consider different timescales inherent to the resource-balancing problem and some remedies for single and multi-user applications. On a long timescale, databases and replicas can be placed to uniformly distribute resource consumption based on historical averages for each database. This mechanism is most useful when the database applications have a predictable load and relatively low variability in resource consumption. Mail databases generally fulfill these conditions, as may many types of corporate applications. An adaptive approach to balancing is built on this foundation of database and replica placement based on historical behavior. The adaptive approach is necessary because historical data is not always predictive of future activity, applications may not have a history, and there will always be fluctuations in application access patterns. Thus, over the course of a day it is often necessary to dynamically redirect clients to the appropriate server replica in order to further balance the load. This can be accomplished by sending a new client to the least loaded replica when the client makes its initial request. The client maintains an affinity to that replica for some moderately long lease period. Short-term surges in load are best handled by hosting multi-client applications in a server cluster. Client requests to an overloaded server are failed over to another cluster member. If the application is not in a cluster, requests are typically rejected, often based on a policy or service agreement that allocates load among different classes of users. The load balancing considerations for this architecture differs considerable from the more widely treated distributed computing model in which applications, or application components are dynamically dispatched to run on remote servers. (See for example Kremien and Kramer 1992, and Wolski et al 1999 and references therein for discussion of load balancing in the DCE framework.)

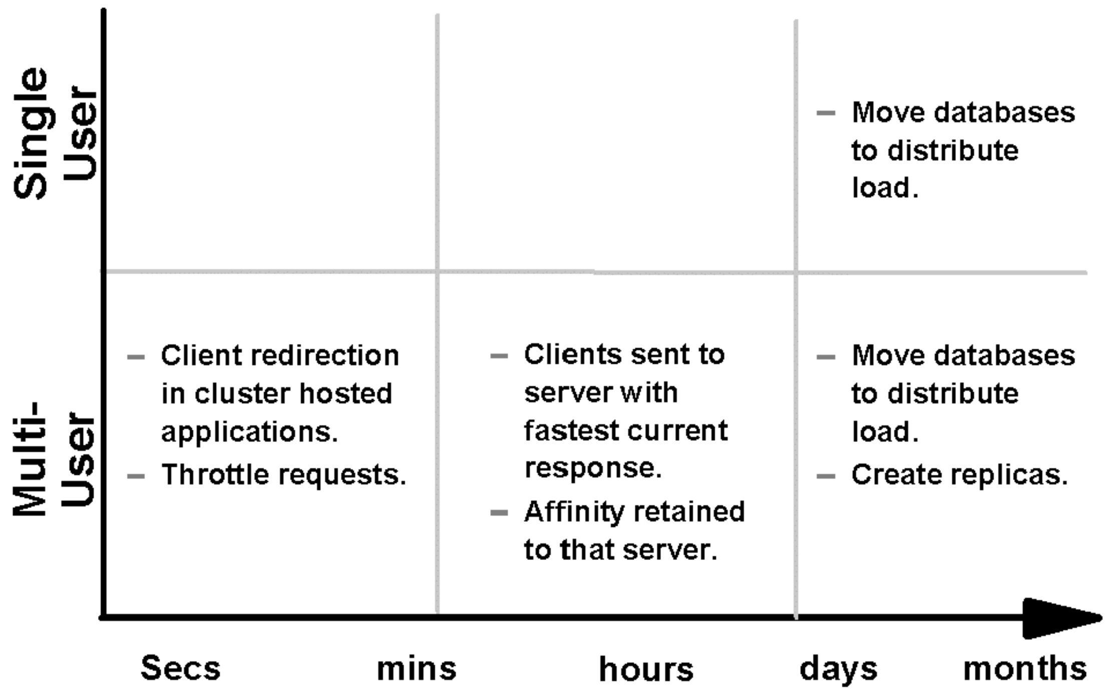
Single User
Multi-User
– Move databases to distribute load.
– Client redirection in cluster hosted applications.
– Throttle requests.
– Clients sent to server with fastest current response.
– Affinity retained to that server.
– Move databases to distribute load.
– Create replicas.
Secs
mins
hours
days
months

Figure1.2. The load balancing problem is addressed in three time domains.

This paper examines the resource management aspects of the architecture of figure 1.1 within the timeframes described in figure 1.2. A second major component of this work is to study some aspects of the concurrent management of several resources. For example, redistributing database applications so that both load and disk space are apportioned according to server capacity. This is not only interesting from a planning and management perspective, but also because an interesting consequence of multidimensional balancing is that it is shown to provide greater stability for load management than simply attacking load. This is discussed in detail in section 3. The analysis is made concrete and quantitative where possible by using the context of the Lotus Notes multi-server application environment. This is an important distributed application environment with over 200,000 users in IBM and 50 million licenses worldwide. Furthermore, Lotus Notes databases fit well into the model of figure 1 and the categories of mail and application databases. We provide a brief overview of the features of Lotus Notes germane to work presented here. More didactic references include Lamb and Lew 1999, and Thomas and Hoyt 1998.

Lotus Notes is a database environment that combines a proprietary on disk data structure with a database management system. The database management system executes as a server task to allow access to the database via a set of proprietary remote procedure calls from a Notes client. Access to the server databases is also provided using several standard protocols such as HTTP for web, and POP3 for mail. The Notes client is packaged with a subset of the server code that enables it to access databases in the client's local file system. The fundamental data construct of a Lotus Notes database is called a 'note'. A 'note' is a semi-structured container consisting of user-defined fields such as rich text and numeric items. No relational structure, in the sense of normalized tables, is imposed on the 'notes' or their items. This loose data structure is useful for groupware and messaging applications (Malone 1987). A Lotus Notes database consists of a collection of 'notes'. The 'notes' are principally organized by and presented to the database management system as views. Views are collections of related notes that can be categorized and sorted.

Lotus Notes has several distinct architectural features that determine its scalability in hosting applications in a multi-server system. Applications are developed for the Notes platform using a built in macro-function and scripting language (Java is also included after release version 4.6). Code written in these languages can be stored in the database as part of its design elements. Thus the Notes application model is to include the data and the code that uses that data in the same database. Database replicas can be created on different servers using a process that is part of the Notes server. The replication process is responsible for synchronizing both the data and design elements of the database. Because Notes is designed primarily for mail and groupware applications in weakly connected networks, strict coherence of database replicas is not required. The replication process merges changes in divergent database replicas and flags incompatible and conflicting updates that must be resolved by a human (Kalwell, Beckhardt et al 1992). The synchronization process can be configured to be push or pull, and can be scheduled or triggered by certain events. The replication algorithm derives from earlier work on synchronized replicas (Demers et al, 1988). The synchronization mechanism facilitates the distribution and maintenance of versions of application code across widely distributed servers. It is also well tailored for intermittently connected networks, mobile users, and 'edge-of-network' servers such as branch offices. At present, Notes does not support transactions. (For a review of 'lazy' synchronization techniques in transactional systems see Pacitti and Simon 2000.) Since application code is typically contained in the database, application mobility is also provided via the creation of database replicas. Consequently, most Notes applications are embedded in the database. (An external database access API is also available in several computer languages to support applications, but this is not the usual approach.).

Accurate techniques of quantifying performance objectives provide the foundation for implementing dynamic load balancing on any timescale. This topic is the subject of section 2. It is argued that expansion factor is an appropriate load metric for a heterogeneous multi-server system. Subsequent sections provide detail on our approaches to balancing load on the timescales outlined above. Section 3 introduces a strategy for the long-term placement of databases based on equalizing the profile of resource utilization on each server. It also discusses how managing multiple resources can improve the balancing and long-term stability of each resource. Section 4 considers adaptive load balancing through the creation of replicas and redirection of client requests. The discussion includes consideration of synchronization costs based on empirical data from a Notes system. The final section returns to balancing multiple resources and presents a heuristic algorithm we developed to optimally place databases and application replicas on servers. Performance of the algorithm is illustrated by its application to balancing resources in IBM's Notes server farms.

2. Performance and Load Metrics

The objectives of load balancing are to optimize utilization of server processing capacity, prevent the server overloading, and provide consistent and acceptable client response times. Server overload occurs when the amount of concurrent work is so large that thread contention and possibly starvation of essential kernel threads prevent progress on any work. However, before this condition is reached, the server response time starts to slow as the internal queues build up. An important consequence of the underlying queuing behavior is that the response times become highly variable. A premise of the work presented here is that it is important to keep the server out of this region of unpredictable service times. This is the case even though on a powerful server the range of responses might still fall within the range of a service agreement. This is because the server is in a state where small additional load can drive it to an unpredictable response time. An appropriate metric on which to base a server load and capacity is the expansion factor (Tetzlaff 1979). The expansion factor (XF) is the actual time to complete a service request normalized to the service time when the request has no contention for the processor or other resources. XF is a generalization of the system time (time spent in the queue plus service time, following the nomenclature of Kleinrock (Kleinrock 1975) that takes into account processing cycles and wait time for resources such as disk and network. In certain operating systems (e.g. IBM system 390) it is possible to quantify the components of XF. For the UNIX and Windows NT servers commonly hosting Notes applications this breakdown is not available. The paper by Wolski (Wolski et al 1999) contains an interesting study of the application of XF and other load metrics to dynamic scheduling in a UNIX distributed computing environment.

It is useful to couch the discussion of XF in terms of the G/M/1 queue. The queuing discipline, arrival and service time distributions underlying a production Notes server are not precisely those of the G/M/1 queue. However, the empirical behavior of the system time for Note's requests closely approximates this theoretical model. This is true particularly in the asymptotic region in which the processor utilization approaches unity and the XF is unbounded. A key result of G/M/1 theory is that the probability that the system time exceeds any particular value is exponentially distributed. This means that XF is parameterized by the mean processor utilization p. Since the queue length behavior in the regime where p->1 for many variations of queues is well characterized both theoretically and empirically (see below) by the M/M/1 result we cite it here (e.g. Kleinrock 1975).

$$XF = 1 + p/(1 - p) = 1/(1-p) \quad (1.0).$$

A real server, including multi-processors, typically implements a more complex management scheme for system resources, queuing discipline, and service times than the simple Markov process used to derive relation 1.0. In fact, it is possible that the XF becomes unbounded even when there is available CPU because some other resource is limiting performance. So it is useful to think of p as an empirical parameterization of the resource used servicing requests.
Perhaps the most important consequence of the exponential distribution of waiting times is that the standard deviation of wait time is proportional to the queue length. Thus, the service time r becomes rapidly unpredictable as p->1. This behavior is demonstrated in the performance data of figure 2.1 taken from a production mail server.

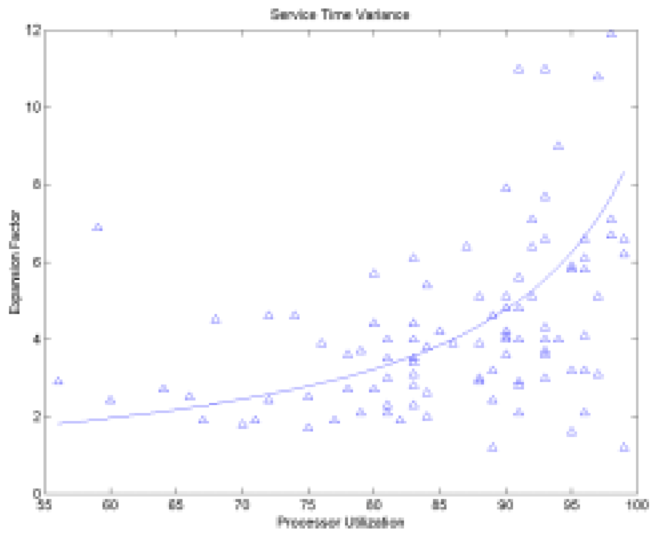

Figure 2.1. Measurements of expansion factor in 15-second intervals as a function of processor utilization on a production Notes server. As XF increases so does the variability in service time.

The server is a single processor IBM RS6000 running AIX, an IBM variant of the UNIX operating system. In Notes, a semantic operation against a database by a Notes client consists of a sequence of remote procedure calls (RPC). Some of these operations are I/O intensive, while others are CPU intensive. The RPC service times are obtained from a log file that can be started on the Lotus Notes server using the SERVER_CLOCK flag. Operating system performance data including CPU utilization and system run queue were obtained by simultaneously capturing the output of the AIX VMSTAT utility in a log file. The service times of the subset of mainly CPU intensive calls is monitored in bins of twenty-second duration. The XF is obtained from the service times by normalizing to minimum response time obtained off shift when the server is lightly loaded. Figure 2.2 shows the distribution of the arrival rate of transactions during the period over which this data was accumulated. The solid line in the figure is the Poisson distribution for the mean of the data. The arrival distribution is somewhat narrower than would be expected from a purely exponential distribution of inter arrival times.

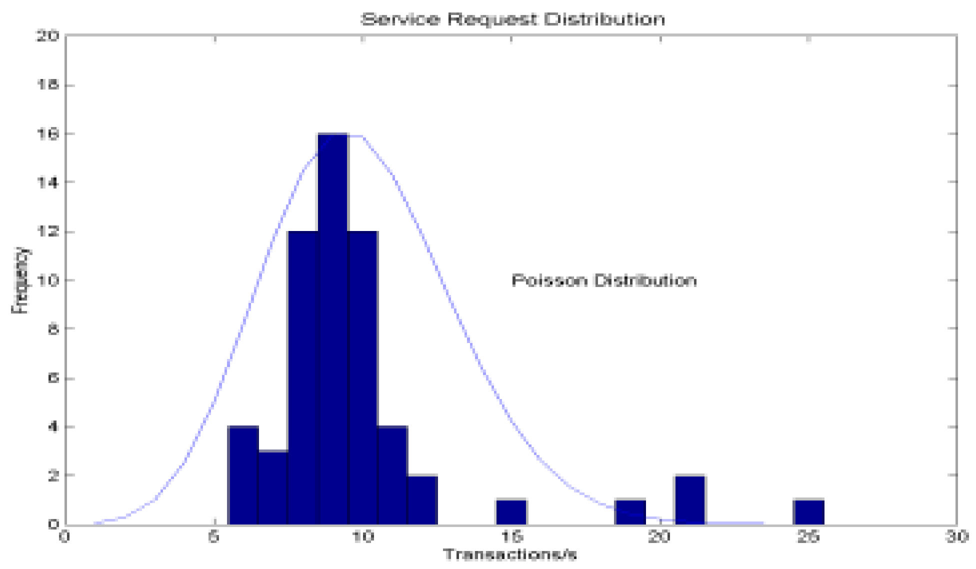

Figure 2.2. Request arrival distribution histogram on a production Notes server. The solid line is the theoretical Poisson distribution having the same mean as the data.

When the CPU utilization approaches unity (p->1) it is more useful to view the service time as a function of the length of the run queue. Figure 2.3 shows XF for the same server against the run queue as reported by VMSTAT.

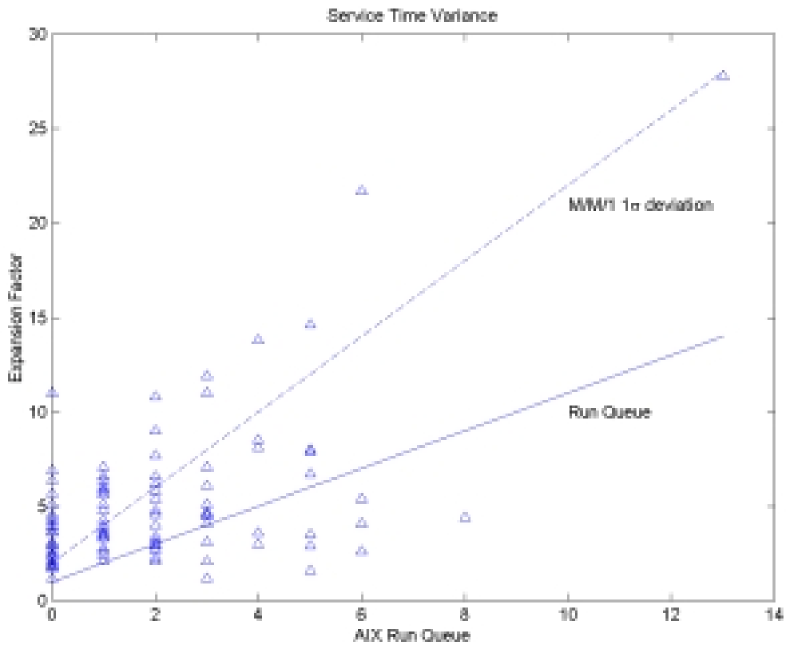

Figure 2.3. This figure shows the observed correlation between expansion factor and system run queue for a Notes server running AIX. The solid line is the run queue average as given by the system VMSTAT utility. The dashed line is the mean queue length plus one standard deviation predicted by the M/M/1 model.

The solid line labeled ‘Run Queue’ indicates the mean XF predicted by the G/M/1 model as in relation 1.0. In other words, the mean XF is simply the length of the run queue. The dashed line is the mean XF plus one standard deviation for this model. As noted above, the ideal G/M/1 model predicts that the standard deviation is equal to the queue length. Thus, the chart shows that at a given queue length, the fluctuations in XF are roughly in agreement with the predictions of queuing theory. The correlation of the uncertainty in run queue with the average expansion factor also indicates that the variability in service time is dominated by contention for the processor. Variability in service times at low values of the run queue (or CPU) indicates contention for locks and system resources for which the process is not considered ready to run by the scheduler.

The relation 1.0 between resource utilization and XF shows that a change in load p -> p + u produces a new XF’.

$$\mathrm{XF}' = \mathrm{XF}/(1-\mathrm{uXF}) \quad (1.1)$$

Where we require that uXF<1. Thus small fluctuations in load drive highly variable server response times at large XF. Regardless of the speed of the CPU, it is reasonable to say the server is at its capacity when XF is somewhere around the knee of the XF (response time) curve in figure 2.1. By using XF to define capacity we have removed the need to directly measure resource utilization. This is important because servers are typically loaded to high average utilizations where a small error in measuring CPU leads to a much larger uncertainty in XF. From equation 1.0

$$\Delta\mathrm{XF}/\mathrm{XF} = (\mathrm{XF}-1)\,\Delta\mathrm{p}/\mathrm{p}\ .$$

For example, at XF=8 a determination of XF to 20% requires measuring p to 3%. Furthermore, XF has a more direct relation to server response time than CPU utilization, which must be translated or calibrated using relations like 1.0. Because the XF is a monotonic function of the resource utilization, a homogeneous ensemble of servers will exhibit similar performance characteristics. The implications of the choice of XF to balance load are best seen in a heterogeneous collection of servers. Each server is limited to run at the same XF. This policy leads to unequal distributions of response times from different servers. At a constant limiting XF’, a server with twice the power will have on average, half the response time. If capacity in terms of XF of the more powerful server is set at twice XF’, the average response times would be the same. However, for the same percentage fluctuation in load, the server running at twice the XF would see a much larger degradation in the predictability of response time.

Consider load balancing amongst an ensemble of heterogeneous servers. Let r_i be the relative power of the i^{th} server, so that r_i indicates the response times of servers when they are unloaded. Using relation 1.1 we can find the relative resource available when the current expansion factor and the limiting capacity are XF and XF’ respectively. The product of r_i with u is the total CPU cycles c_i available on the i^{th} server. This leads to,

$$c_i = r_i[(\mathrm{XF}' - \mathrm{XF}_i)/\mathrm{XF}_i\mathrm{XF}'] \quad (1.2).$$

For example, consider two servers having a current load of $XF_1 = 2$, $XF_2 = 3$, and assume the limiting XF is set by policy at XF’ = 5. Furthermore, the servers have relative powers $r_1 = 1$ and $r_2 = 2$. Then $c_1 = 0.25$ and $c_2 = 0.17$. Suppose a system workload manager receives a request from a busy customer who will need to average 0.01 cycles. Although the job can be handled on either machine, it would be sent to the slower one. The c_i indicate the proportion to which a fixed amount of load should be assigned to the server. The above example applied a deterministic policy of assigning work to the server with the lowest XF. If the load balancing is carried out by a stochastic front end, the probability of sending load to the i^{th} server would be given by the normalized c_i. An example of such a front end application that could use XF either stochastically, or as a deterministic load measure would be IBM's network dispatcher for routing http requests (Goldszmidt and Hunt 1999).

Finally consider some practical implications of measuring expansion factor. The discussion so far has proceeded couched in the language of the M/M/1 model. The service time data presented was obtained by offline reduction from the Notes server log and was correlated to simultaneous operating system data. Clearly it would be useful to collect this data in real time in the Notes server. Because the Notes server is a cross platform application, integration of operating system data into the OS abstraction layer of the code is a complex design and programming task. However, it is possible to make a reasonable estimation of XF using heuristics available within Notes and the proportion of the CPU allocated to this process. The Notes server uses pools of threads to handle server requests. The thread pool runs under a single process at the operating system level. When requests back up because of contention for any resource, the number of concurrent active threads increases. When other processes such as HTTP are executing, the XF estimated by the number of active RPC threads is scaled by the CPU utilization of the process servicing RPCs. A proxy for the XF is the number of threads concurrently servicing RPC requests normalized to the CPU for this process. To see this note that XF is the ratio, R/s, of response time to service time. Multiply the numerator and denominator by the arrival rate and apply Little's result (Kleinrock 1975) to show that the numerator is the number of concurrent threads and the denominator is the fractional utilization (Hellerstein 1999).

Therefore, server capacity is a limiting value of the XF at which the machine response is considered predictable. Because one phase of our load balancing solution requires distributing databases according to their load, we now need to apportion the server load at the database level. One candidate metric is CPU cycles. This requires accumulating the CPU cycles against each database. Accounting for CPU at this granularity is not generally possible. This is because a thread pool is used to service RPC's, and very few operating systems accumulate CPU on a per thread basis. Microsoft Windows NT and IBM OS/390 are two exceptions, while POSIX compliant operating systems are not required to do so. A more practical solution is based on the number and frequency of RPCs. The Notes server logs this information at the server and database granularities. The RPC rate at the server level can be calibrated against the observed XF so that the limiting XF is expressed in units of RPC rate. This is the approach taken here, but first we note some potential difficulties. The RPC rate is summed over all requests, and there is great variability in the type and amount of server resource consumed by each RPC. Ideally, we could select and monitor one or more subsets of RPC that are known to be, for example, CPU intensive. One such subset was used to produce the data or figure 1. However, presently such analysis is only available offline. So the RPC based algorithms described in the paper work best when balancing within a class of databases having a similar distribution of workload. Fortunately, a very large and important category of this type is mail databases. Also it is common to keep mail databases on servers dedicated to mail. So the assumptions that total RPC traffic is correlated with load works reasonably well for a very important class of databases. General application databases such as collaborative databases or catalog purchasing exhibit much more variable access patterns and workloads. For these databases a placement strategy based strictly on total RPC count is not as reliable. Future versions of the Notes server will have to be instrumented to provide more direct measures of resource consumption for these databases – at which time the technique described in Section 3 can be used for all database types. An interim solution based on an administrator creating a replica combined with client redirection can provide some dynamic load balancing in this regime. As noted in the introduction, the ability to dynamically create a replica would provide an additional degree of freedom in addressing this problem. Replica creation is discussed in more detail in section 4.

3. Database Placement Using Equal Load Profiles

This section investigates the problem of statically placing databases and their replicas based on historical usage statistics. Databases exhibit variable load both due to the statistical nature of being accessed by individual clients, but also because of systematic behavior of client accesses. The latter is especially true of mail databases where a single client dominates usage. Two databases with the same average utilization may exhibit very different access patterns (Pope 1998). Some mail clients may work intensively for a day and be out the next day, while others work consistently. The former will have a much greater standard deviation in load than the latter. Clearly we would like to avoid putting all the databases with highly variable access patterns on the same server. So in addition to balancing server load, we would like a placement strategy that minimizes the variability in load.

Fluctuations in load on each server are measured by the coefficient of variation in activity. The coefficient of variation is the ratio of the standard deviation to the mean. A mail server typically hosts hundreds of mail databases. Consequently the coefficient of variation for the server is considerably less than for the individual databases. The coefficient of variation averaged over a group of servers is most likely to be minimized when both the total resource load and the profile of databases is equalized. For example, two servers of similar capacity can be load balanced by placing a few of the most active users on one server and many less active users on the second server. In the case of real users who don't have stationary, exponentially distributed access patterns such a solution will typically produce greater fluctuations in load and response on the server with the few largest users. One might expect that improving the overall symmetry by equalizing both the load and the profile of load would lead to a stable solution. As a simple example consider N databases to be distributed on two servers. The N total databases consist of two subsets; N_1 databases with average activity A_1 and N_2 databases with activity A_2. Assume the average daily activity against the databases has an exponential distribution. Then the daily activity variance of the i^{th} subset is A_i. Define the coefficient of variation V_i for a server as the ratio of standard deviation to load so that $V_i^2 = N_i A_i^2 / (N_i A_i)^2 = 1/N_i$. Assign values $(N_{1,} A_1)$= (4,10) and $(N_{2,} A_2)$ = (10,4). When load is balanced by placing each subset on a separate server the squares of the coefficients of variation are (0.25, 0.1) for the given values. In contrast, choosing the symmetric distribution of databases in which the subsets are apportioned evenly between the servers produces a coefficient of variation of 0.175 on each server.

In the above example, we assumed the database populations have well defined and measured statistical variability. In reality, it is not always possible in practice to obtain measurements of the usage patterns on each database. And it may be impractical to individually characterize them by statistical measure since the underlying pattern of client access is highly systematic. However, a reasonable heuristic approach is to use information contained in other attributes of the database. For example, mail databases with similar mean values of attributes such as activity, disk space, and rates of bytes written or read might be expected to have similar variability in these attributes. The more attributes considered, the greater the similarity one would expect. This is an approximation. As noted above, a detailed study of Notes mail databases shows the presence of a more complex taxonomy of daily user access (Pope 1998). In practice, however, this assumption works well as demonstrated by data presented later in this paper. It also fits in well with another objective of this work, which is to balance and manage multiple resources or attributes. In fact, in addition to balancing total resource on the server, we apply this principle of equalizing the profiles to each of the resources being managed. These dual objectives are better explained with reference to figure 3.1.

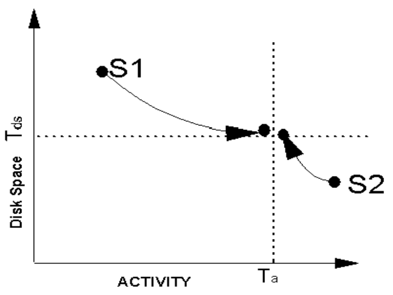

Fig 3.1a

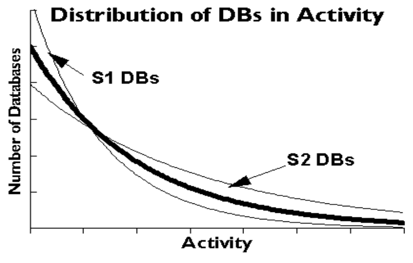

Fig 3.1b

Figure 3.1. Dual objectives of balancing total load and disk space on two servers (a) while also equalizing the database profiles (b). In (b) the heavy solid line is the activity distribution of all databases on both servers. The lighter lines indicate the initial distributions of the databases on S1 and S2. After balancing, they should be more similar to the heavy solid line.

Figure 3.1a shows two servers, S1 and S2 with initial coordinates reflecting their total values of activity and disk space. New targets based on their capacities are achieved by redistributing the applications so that the servers move to locations near the targets. Figure 3.1b shows the activity distribution of all databases on S1 and S2 combined. We have assumed an exponential distribution (this is quite typical for real applications, see data in section 5). The example of figure 3.1b also shows that S1 starts with an excess of low activity databases, and S2 and excess of high activity databases. The concept presented here is that in addition to reaching the target goals of figure 3.1a, we would like the profiles of figure 3.2b to also be equalized. Of course, it is not always possible to achieve this balance because mathematically the problem is over constrained. But in practice, with the typically large number of databases (production servers generally host more than 500 databases.) hosted on a real system it can closely approached. We have developed a heuristic algorithm to achieve this goal, but defer its discussion until section 5 in order to maintain continuity with the more general treatment of load balancing.

The example just provided of balancing two classes of databases suggests that the approach of balancing total load, and also equalizing the distribution of databases might reduce the worst-case coefficient of variation among the set of servers. The basis of this insight about increased symmetry is rooted in the following more general argument. Consider how balancing the distribution of databases as well as the total activity minimizes the variability in daily activity. To simplify the notation, the argument is developed using two resources, activity (α) and disk space (β). Databases are to be placed on M homogeneous servers so that the activity is equalized. By assumption, databases with attribute coordinates (α,β) have similar access patterns. The number density of all databases in the system of M servers falling within a small region of coordinates (α,β) is described by the density function $n(\alpha,\beta)$. $n(\alpha,\beta)$ is the sum of the distributions $n_i(\alpha,\beta)$ on each server. The average activity load on the i^{th} server is:

$$\overline{\alpha}_i = \quad \alpha n_i(\alpha, \beta) d\alpha \, d\beta \, . \qquad (3.1)$$

Suppose that databases with coordinates (α, β) are characterized by a variance in activity of $\sigma_\alpha^2 \approx \sigma_\alpha^2(\alpha, \beta)$. Then the activity variance on the ith server is

$$\sigma_{\alpha(i)}^2 = \quad \sigma_\alpha^2(\alpha, \beta) n_i(\alpha, \beta) d\alpha \, d\beta \, . \qquad (3.2)$$

The coefficient of variation on the ith server is given by $V_{\alpha_i}^2 = \sigma_{\alpha_i}^2 / \overline{\alpha}_i^2$.

The load balance condition $\overline{\alpha}_i \approx \overline{\alpha}_j$ for $i \neq j$ should be met subject to the constraint $\sigma_{\alpha(i)} \approx \sigma_{\alpha(j)}$. This will equalize the coefficients of variation on the servers. When a Poisson distribution describes the activity rate at each database, $\sigma_\alpha^2 \propto \alpha$, the variance condition is satisfied so long as the load is balanced. More generally relations 3.1 and 3.2 require that a balanced solution satisfy the following conditions for servers $i \neq j$.

$$\alpha \left[n_i(\alpha, \beta) - n_j(\alpha, \beta) \right] d\alpha \, d\beta \cong 0 \qquad (3.3)$$

$$\sigma_\alpha^2(\alpha, \beta) \left[n_i(\alpha, \beta) - n_j(\alpha, \beta) \right] d\alpha \, d\beta \cong 0 \qquad (3.4)$$

The most general solution to relations 3.1 and 3.2 occurs when

$$n_i(\alpha, \beta) \cong n_j(\alpha, \beta) . \qquad (3.4)$$

Thus, as long as the database access pattern is correlated with the resource attributes the condition of equal density functions minimizes the maximum of the coefficient of variation across the servers. Another consequence of improved symmetry is that long-term rates of growth (or decline) in resources tend to be equalized across the servers. This is evident by replacing the activity fluctuations in the above analysis with any other property whose value is associated with the database parameters. In practice the database density function $n(\alpha, \beta)$ is usually exponentially distributed with the average resource use. The functional behavior of the density function for mail databases is discussed fully in section 5 of this paper, which presents a heuristic algorithm used to achieve condition 3.4

We mention some of the practical consequences of balancing servers to uniform distributions of databases. The most important is that the relative proportions of resources on each server tend toward that for the aggregate collection of all databases. For example, consider balancing five heterogeneous servers with an aggregate RPC rate of 5 million/day and a total of 100GB of disk space available for Notes databases. This means that the aggregate has a relative proportion of 20KB/transaction. Uniformly populating the servers requires this ratio on each server. From a capacity planning point of view this means that the server should have disk space and processing power to support this ratio. The server could have an excess of either, and the target load for that server would be determined by the most restrictive capacity, either processing or disk space. In practice, the heuristic algorithm described in section 5, which we developed to perform balancing, approximates the continuous distribution using a small number of discrete bins. This approximation allows administrators some flexibility in specifying individual server targets.

The consequences of equal balancing proportions extend to the rate of growth of resource consumption. A concept commonly used in capacity planning and forecasting is the time before a resource expires. Pope has emphasized the importance of lifetime and growth rates in disk management and transaction capacity planning and applied lifetime goals to the management of Notes servers (Pope and Mummert 1999). We briefly explore the consequences of the approach taken here on growth rates. Because similar distributions of databases are placed on each server, the ratio of resource growth rate to resource occupancy on uniformly balanced servers also tends to the value for the aggregate. We make the observation that for a given collection of servers and databases it is possible to make the time before additional resource is required all servers the same as that for the aggregate. To make this more clear, denote total resource for the aggregate of all databases by R, and the rate of change by R'. The total system capacity is C. Under the assumption of constant linear growth, the time before more capacity needs to be added somewhere in this system is $T = (C - R)/R'$. The corresponding time on the ith server is $T_i = (C_i - R_i)/R_i'$. A uniform distribution of databases according to (3.4) requires that $R/R' = R_i/R_i'$. So if the resource capacity of each server is balanced in the proportion to the aggregate (i.e. from the example above, 20KB of disk space for each RPC so that a server capable of handling a million RPCs will have 20GB of disk space). The time to exhaust the resources on any one server is the same as the time for the entire collection of servers. The point is that the lower bound on the period at which any one server reaches overload can be made equal to the time at which there is not enough resource in the entire system and capacity must be added anyway. We do note that there are many other important factors in capacity planning such as the costs of managing, purchasing, and adding resources. For example, if the cost of disk space is dropping 50%/year it may make more sense to upgrade capacity slowly. Consequently, economic factors and business policies often play as much of a role in planning as the above technical considerations.

It is also important to stress that the symmetric balancing across servers is not optimal for all load-balancing problems. In fact there are situations in which it is desirable from a performance viewpoint to concentrate databases with similar resource use on the same server. An example is the placement of files for interactive video on video servers (Dan and Sitaram 1995). In this case, placing databases with similar ratios of size to access rate on each server optimizes the performance criteria. Other types of asymmetric balancing may be justified when different servers have been tuned for different types of workload. Another example is in the placing of a known workload distribution on collection of servers. Mor-Hecht has studied optimum partitioning of the long tailed workload distribution in which all jobs within a certain size range are sent to a dedicated server. This work uses also uses XF as an optimization criterion. In a Notes environment this approach corresponds to the situation in which dedicated servers are used for mail because mail databases exhibit similar workloads and administrative expertise. As noted above, these considerations are often just one input to the complex problem of managing a distributed, heterogeneous, computer system.

4. Adaptive balancing by replica creation and client redirection

The activity that applications place on servers varies from that of electronic mail databases in the following ways. While it is extremely unlikely that a single mail database could exceed the capacity of a single server, it is not unusual to have applications that require multiple servers in order to have reasonable responsiveness. This can be achieved via the creation of replica(s) on another server(s) and the scheduling of synchronization to occur at a time interval based on the application’s requirements for data consistency (e.g., immediately, hourly, daily, etc.). Furthermore, the coefficient of variation of the daily activity of a mail database tends to be relatively small. While this is also true of many applications, there are a significant number of applications that exhibit cyclical activity patterns (e.g., an application that is only active the last few days of a month) and others that exhibit predictable (e.g., Christmas sales activity) or unpredictable surges of activity. Often, a server administration team is unaware of imminent activity due to variations of even known cause (i.e., to someone in the organization).

One strategy for adaptive balancing of applications is to execute a placement algorithm (as described above) more frequently than would usually be done for mail databases. This might be daily or several times a day depending on the nature of the applications. The algorithm could also be triggered by some event such as a service threshold exceeded. Instead of reorganizing large sets of applications (which might be required initially), the algorithm would typically run in a tune up mode where it was looking for unusual increases or decreases in activity.

However, active management of database replicas provides a complementary and powerful approach to handling load variations on these intermediate time scales. There are two components to this approach.

1) Creating replicas when an application requires more resources than are currently available on a given server. Removing replicas when activity diminishes to the point where they are no longer required.
2) Redirecting clients to the available replicas to exploit the increased resources.

Creating a new replica offloads a portion of the client read and write traffic, but adds the cost of propagating and receiving writes to and from the other replicas. We studied the replication mechanism and measured the update cost between two Lotus Notes databases. After presenting these data we provide general guidelines for when to create a new replica.

Subsequent to replica creation, clients must be directed to the replicas in a way that distributes the load, prevents surges from all clients rushing to the momentarily least loaded server, and does not require a complex central management facility. A solution based on client side sampling of replica response times is described in section 4.2. Monte Carlo simulations are used to demonstrate the feasibility of this technique.

There are other considerations in the placement of replicas including link bandwidth and the cost of file storage. The problem of optimizing the number and location of file copies on a heterogeneous network subject to delay and availability constraints has been addressed in the literature. Mahmoud and Riordon provide an excellent discussion of the problem and offer exact and scalable heuristic solutions to this file allocation problem (Mahmoud and Riordon 1976). The work here is complementary, focusing on server performance, and attacking the problem of directing clients to server replicas so that uniform load is achieved with low variability.

4.1 Replica creation, overhead, and cost

Lotus Notes database replication consists of several sequential phases whose time complexity is difficult to predict in advance. The first step consists of determining the replication history between the source and destination. One product of this history is a list of the database documents that need to be replicated. The creation of the replication history can be a relatively high overhead portion of the entire replication process and so the history results are cached. Replication events that result in a “hit” in the replication history cache proceed significantly faster than if the replication history has to be built from scratch. In addition, there is a cache of open databases (databases are not closed when a “close” event occurs but kept in a cache and only closed when they age out of that cache). If the replication event is for a database that is in this cache, then the process is further expedited.

These effects make it very difficult to predict replication overhead for any specific database on a set of servers. In order to attain a reasonable estimate of the replication overhead an empirical analysis was undertaken. We did replication experiments by varying document sizes and the number of documents replicated in each batch. We also measured the effects of the database and replication history caches. It became quickly clear that the variables involved had a significant effect on the response time and that recommendations based on our results would be rules of thumb at best.[1]

Replication in Lotus Notes is either "pull" (the initiating system reads data from the target system) or "push" (the initiating system writes data to the target system). The initiating system controls the replication event and uses the standard set of RPC's to open, read/write, close, etc. the remote database. In order to measure the total overhead involved in replication (i.e., on both systems) corresponding sets of experiments were added. That is pull/push overhead measured from the puller/pusher was added to the overhead seen on the "pullee/pushee". Although the individual components varied, the sums were similar for both pushing and pulling. The values given here are overheads seen when replication was in "pull" mode.

The primary goal was to understand the components of replication overhead sufficiently to be able to evaluate the benefit of replicating a given database to another sever(s) in order to balance load. We wanted to measure the components of the replication overhead in such a way that they could be expressed as ratios of common events such as document creation and document reading. This would tend to normalize the results so they could then be used to make decisions using hardware infrastructures other than that used for the measurements.[2]

While we wanted to derive the overhead of replication in relation to both the overhead of the actual document creation event and the read event, we wanted to measure other aspects of replication that would affect our load balancing strategy. We were especially interested in any batching effects of replication, i.e. whether the ratio of replication overhead to document creation overhead diminished as the number of documents replicated increased. In addition, we wanted to measure the amount of overhead contributed by the size of the document.

There is a significant caching effect in replication, both from the data itself (avoiding local I/O) and especially for the replication history that is stored in B-trees. We witnessed long replication startup times when there was no history cache.

In addition, the overheads would vary, usually not in the same day, but over weeks there would be significant variation with no obvious reason. When the values would change they would typically be repeatable, indicating some significant change in the system response and not a transitory occurrence. This is an interesting area for further research as it may indicate some fundamental performance issue such as memory fragmentation that could be resolved. We constantly recalibrated system idle overhead, and although this occasionally varied, it was not the cause of these fluctuations.

For our replication measurements, the very long startup times were discarded because they were not quantifiable in terms of frequency, probability, etc. Therefore our measurements are likely to represent a lower bound on the replication overhead and would only be likely to be approached

[1]A better long-term solution would be to modify the replication engine to maintain, for each object that is replicated, statistics for each separate target replica. This should integrate the dynamic effects of caching, network topology, etc. At the time this research was done we did not have the ability to modify the source code, so this was not a viable option.
[2]It is not clear however that measurements for a given release of Lotus Notes would be valid for subsequent (or earlier) releases. This implies periodic reevaluation of the results, if this type of empirical analysis were to be used.

on systems with relatively high activity and frequent replication. However since these are the most likely scenarios for load balancing to be useful, we believe this is a reasonable approach.

Measurements

We created an agent that ran on a Lotus Notes client and automated the creation of multiple documents. We ran tests with null documents, and document sizes of 1 KB, 6 KB, and 32 KB.[3] For each size we measured the replication time for 1 document and batches of 10, 100, and 500 documents. In addition, we measured the smallest and largest times to read and write a single document. Of course, these are unlikely to be the absolute minimum and maximum times (although the smallest number is probably close to the lower bound) but we believe them to be reasonably representative. The data are presented in table 4.1. The shortest times are achieved by repeatedly reading/writing a document from a database that is already open and in addition to avoiding open/close overhead, this process benefited (at least in the read case) from cache effects. The minimal write overhead might be realized by an agent posting data to an open database and therefore be representative of real-world activity, but the low read overhead would not appear likely to be sustained over any significant period of time. The highest values occurred when a database with a significant number of documents (e.g., 1000) was opened without recent reference, incurring the overhead of building a view, etc. without benefit of cache hits. Expected values will likely vary between these extremes and depend upon database characteristics, reference frequency, and user reference patterns (e.g., do they leave the database open on their desktop). Values obtained by occasional opening and closing and some degree of cache benefit are approximately midway between the extremes seen – and this is probably a reasonable estimate for use in load-balancing calculations involving replication.

Table 4.1

Number of documents	Document size in bytes	CPU time(s) (sum of both systems - e.g., push + pull)	Range of single document read CPU time (s)	Range of single document write time
1	null	2.8	0.01-1.9	0.12-2.0
10	null	2		
100	null	4		
500	null	14		
1	1K	1	0.02-1.6	0.13-1.7
10	1K	3		
100	1K	9		
500	1K	31		
1	6K	1	0.08-1.84	0.28-2.0
10	6K	6		
100	6K	14		
500	6K	20		
1	30K	1	0.03-3.0	0.44-3.4
10	30K	4		
100	30K	17		
500	30K	73		

[3]The null document had a null subject and body but still contained 39 bytes. We were unable to create messages greater than 32 KB with an agent.

Note that there are a few anomalies in this data. The null message time for 1 document exceeds the 1-document times for all other sizes. Also the 500-document batch time for documents of 1K significantly exceeds the time for the same batch of 6K documents. These were repeatable. They may be due to the relationship of document size to communication buffer sizes, or some other adverse internal interaction. At any rate, these anomalies should caution against any rigorous application of the results.

Despite these anomalies, the data of table 4.1 clearly show that the replication cost per document exhibits a strong batching effect. The CPU time per document for 100, 6Kbyte documents is 0.14s, compared to the average range of a single read or write which is of the order of one second.

Using the Replication Data to Estimate Replication Overhead

Lotus Notes allows administrators to configure replication as either peer-to-peer or via a dedicated hub replication. Peer-to-peer replication is usually easier to administer than hub. It is most useful when the write/read ratio is very low, or the data synchronization requirements allow updates to be propagated off shift. This avoids the clearly poor scaling of this configuration. Peer-to-peer may be useful in guaranteeing update propagation for intermittent network connectivity (Demers et al 1988). When near continuous updates are required, peer-to-peer can still be useful when the write/read ratio is low and database availability is a significant consideration. An important example of this latter application is the Lotus Domino server cluster. A set of servers in close physical proximity is configured for peer replication with immediate propagation of updates.

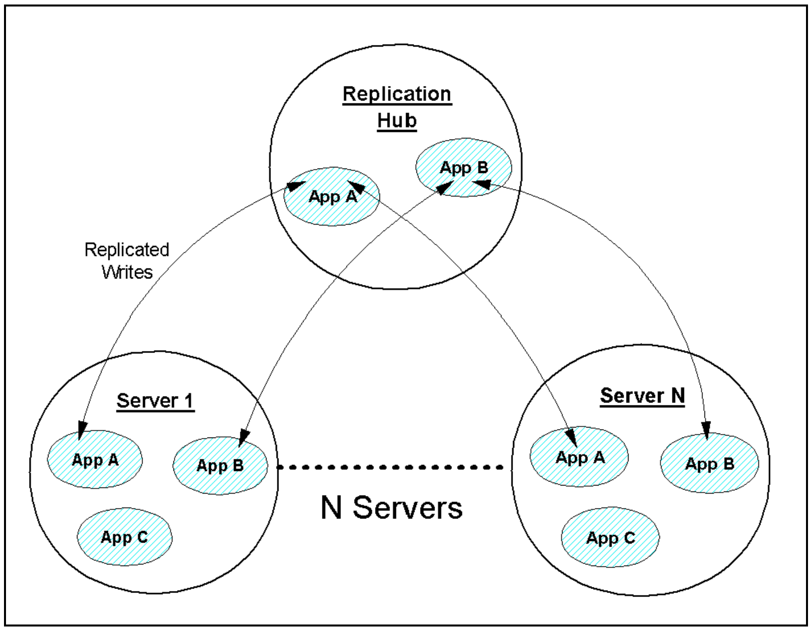

Figure 4.1. Data synchronization between replicas using a dedicated replication hub.

Hub replication is illustrated in figure 4.1. A dedicated replication hub server updates applications' A 'and' B'. Clients do not access the replicas on the hub. This configuration provides good scalability for the interesting case where frequent data synchronization is required and there is a significant proportion of database writes. A simple example illustrates the change in load on an application instance when N replicas are created. We use hub replication and assume frequent update propagation so that the updates can be treated as part of the average server load. Take R and W to be the total number of application read and writes. A single write costs an amount 'a' in

CPU relative to a read. The total accesses are normalized so that R+W=1and the ratio W/R is denoted by r. When there is a single instance of the application in the system the load, L_1 , on the hosting server is proportional to: $L_1 = (R + aW) = (1 + ar)/(1 + r)$. Consider the creation of N instances of the application on N servers using hub replication of figure 4.1. We assume that the client accesses have been approximately uniformly distributed using an approach such as that described in the next section of this paper. Each replica must copy its write load to the replication hub, and receive updates from N-1 other replicas. The application load on any server is related to L_1 by

$$L_N = L_1 - L_1(N-1)/N + [C(W/N) + C((N-1)W/N)].$$

The terms C() in square brackets are the CPU cost of propagation to and from the hub respectively. If replication is frequent so that less than 100 documents are updated each event, we can use the data of table 4.1. The replication cost in column 3 of table 4.1is the cost for both push and pull. Furthermore, the cost can be scaled to another server of different power based on the average single document read and write times of columns 4 and 5. For example, assume the updated documents are in the 6Kbyte size range, and the replication interval is chosen so that on average, 100 documents are batched for update at each execution. For two replicas hosted on servers like the one in the study, the total cost (the sum of both terms in [] in the above expression) is approximately 14 seconds of CPU. If the write/read ratio is 10, then 1,100 seconds of CPU are used on both machines. So creating the second replica has offloaded about 500 seconds of CPU arising mostly from reads at the expense of a 14 second replication cost to update the writes. Of course the replication event causes a short-term load. A future enhancement would be to trigger replication events during local usage minima about a normal schedule. The time and depth of a local minimum would have to be predicted, and there is some ongoing work in this area (Hellerstein, Zhang, and Shahbuddin 1999).

This is admittedly not a precise calculation, but in the absence of several exact values (e.g., number of documents read/written, replication intervals, mean document read/write time) it should provide a reasonable approximation for purposes of deciding whether to create additional replicas of a database to improve the overall balance of load.

4.2 Client Redirection

Once multiple database replicas are available, clients must be redirected to efficiently balance the load. It is interesting to compare the relative performance of the most basic techniques available to perform redirection. Lotus Notes implements a redirection component as a feature of product Release 5 of the client and server code. We compare five basic redirection methods, including that of Lotus Notes Release 5, using a Monte Carlo simulation model developed by D. H. J. Epema (Epema 1999). In this study, a set of N total databases is distributed over M servers. The databases are accessed by a large number of clients. A subset of the databases is replicated across the set of servers and the clients can connect to any of these replicas. The results are not especially sensitive to the composition of this subset. It is only important that the replicated databases contain a sufficient fraction of the total server load to allow load balancing. For example, suppose the servers start out with a load imbalance of 20%. If the replicated databases account for only 10% of the load, balancing cannot be accomplished by client redirection. Client request rates and corresponding server execution times are generated from statistical distributions based on data obtained from production IBM Notes servers. The logical transactions generated by the simulator are strings of individual RPCs whose length and variance are based on data from IBM production servers. Since we wanted to study the effectiveness of client redirection independently of factors such as database read/write ratios, the overhead of replication itself was not considered. We assumed replication intervals and read/write ratios that would generate relatively insignificant overhead. This is true of many real-world situations.

The redirection algorithms used in the simulation are listed below.

1. Better of 2 random servers. The client picks two servers at random from the set that host the database, determines the load on each, and selects the server with the lowest instantaneous load (queue size). This is based on work done by Michael Mitzenmacher in his Ph.D. thesis at Berkeley in 1996 (Mitzenmacher 1996).
2. Client-centric. The client queries all servers that host the database and selects the server with the lowest instantaneous load.
3. Redirection manager. The Redirection Manager is a server task that polls all servers periodically for their current load. The client sends a request to the Redirection Manager, which returns the server with the lowest-load. For the simulations in this paper the poll interval is 60 seconds.
4. Client random. The client selects a server randomly from the set that hosts the database.
5. R5 Product. The client selects the first server from the set that hosts the database that has not marked itself unavailable due to excessive load (i.e., a specified resource utilization factor). We varied the excessive load parameter and found that it had little effect on the results.
6. None. The clients always select the primary replica. Included to establish a worst case.

Simulations are run using each of the above redirection methods including 6) for comparison. A client initially connects to a database replica using the redirection method chosen for the simulation. An affinity to this replica is maintained until the client happens to be idle for an interval that is a simulation parameter. This period of quiescence causes the redirection method to be reapplied when the client becomes active again. This waiting interval should on average exceed the database replication period in order to avoid anomalies due to loose data consistency. Intervals from 15 minutes to an hour yield similar results. Long intervals tend to encumber the selection methods by not permitting adequate server reselection. The simulation was run with two scenarios: 400 clients, 40 servers, 50 databases, and 100 clients, 6 servers, 30 databases. The processing power of the servers was adjusted to yield a reasonable mean CPU load between 60%->80%. Throughput rates in the two simulations varied due to the different server configurations, relative processing power, and utilization. The results are presented in graphical form in figures 4.2 and 4.3. The figures show the response times as shaded bars, and the transaction rate per replica as a solid bar. The response times and transaction rates are normalized to the poorest response time, and highest transaction rate for redirection methods 1-5 above. These are the methods in which there are multiple replicas. The no redirection response time (method 6 which has a single database) is provided for comparison. This response is off scale in the simulation of figure 4.2 because channeling all requests to a single database created a long service queue.

The “better of 2 random servers” and “client-centric best” are the two best methods. They are essentially equivalent in the 6-server scenario and the “better of 2 random servers” has a 5% better response time in the 40-server scenario. This is due to less time spent on server load queries. Both have standard deviations of response time and throughput an order of magnitude smaller than those for the R5 product. Also, it is interesting that the Redirection Manager method is not as good as client-centric until the poll interval becomes very small. We had thought that the Redirection Manager would be more scalable than client-centric – thus justifying the additional software – but with the number of clients and servers we were able to simulate this was not evident. With very large numbers of active clients it very well may prove to be more scalable. However, our study shows that based on simplicity, performance, and robustness “better of 2 random servers” is a good choice for the client redirection method.

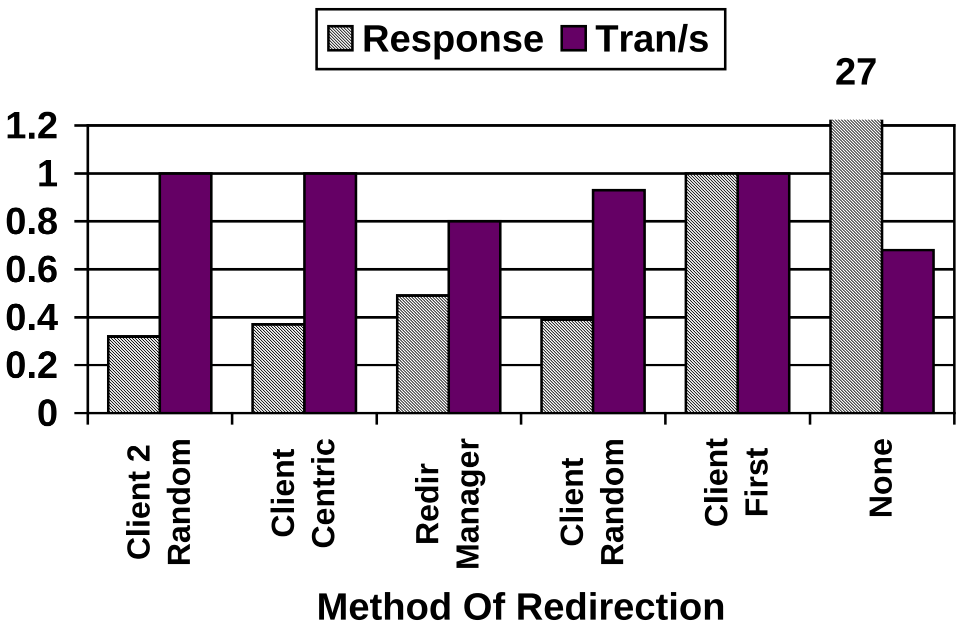

Figure 4.2. Simulation results for the six methods of client redirection discussed in the text. This simulation was performed with 400 clients, 40 servers, and 50 databases. The shaded bars are the normalized response times for each of the methods. The solid bars are the resulting transaction rate.

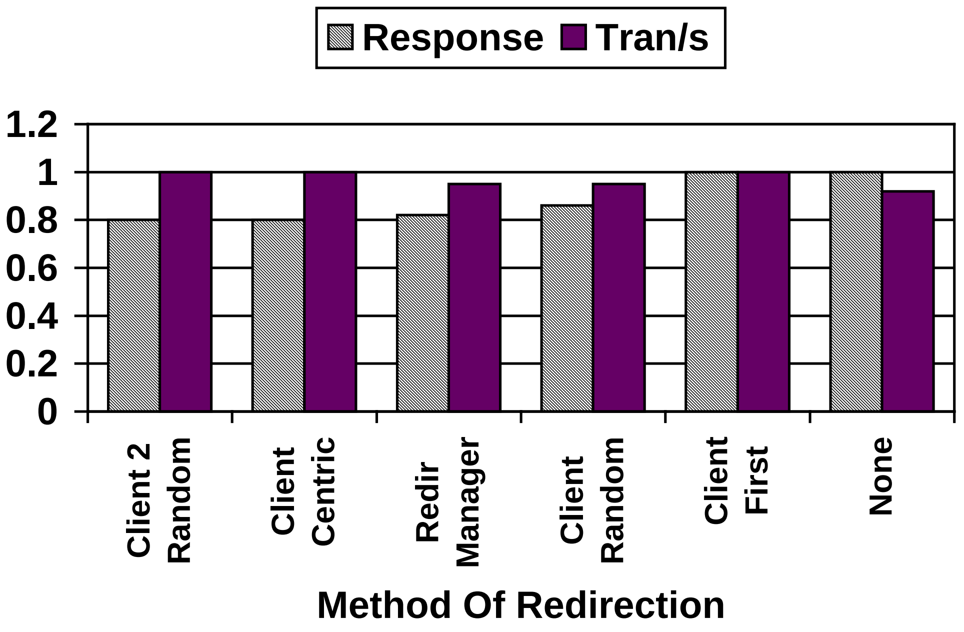

Figure 4.3. Simulation results for the six methods of client redirection discussed in the text. This simulation was performed with 100 clients, 6 servers, and 30 databases. The shaded bars are the normalized response times for each of the methods. The solid bars are the resulting transaction rate.

5. An algorithm for placement of databases

The objective of database placement is to equalize server resource consumption normalized to the capacity of each server, while achieving a similar profile of databases on each server. The database profile is described by the joint density function $n(\alpha, \beta, \ldots)$ for all databases across the servers being balanced. The parameters of the density function are the resources to be balanced such as transactions per day, disk space size, or bytes written. A typical density function for mail databases is indicated in the scatter plot of figure 5.1. The resources used to parameterize this density function are transactions/day and disk space.

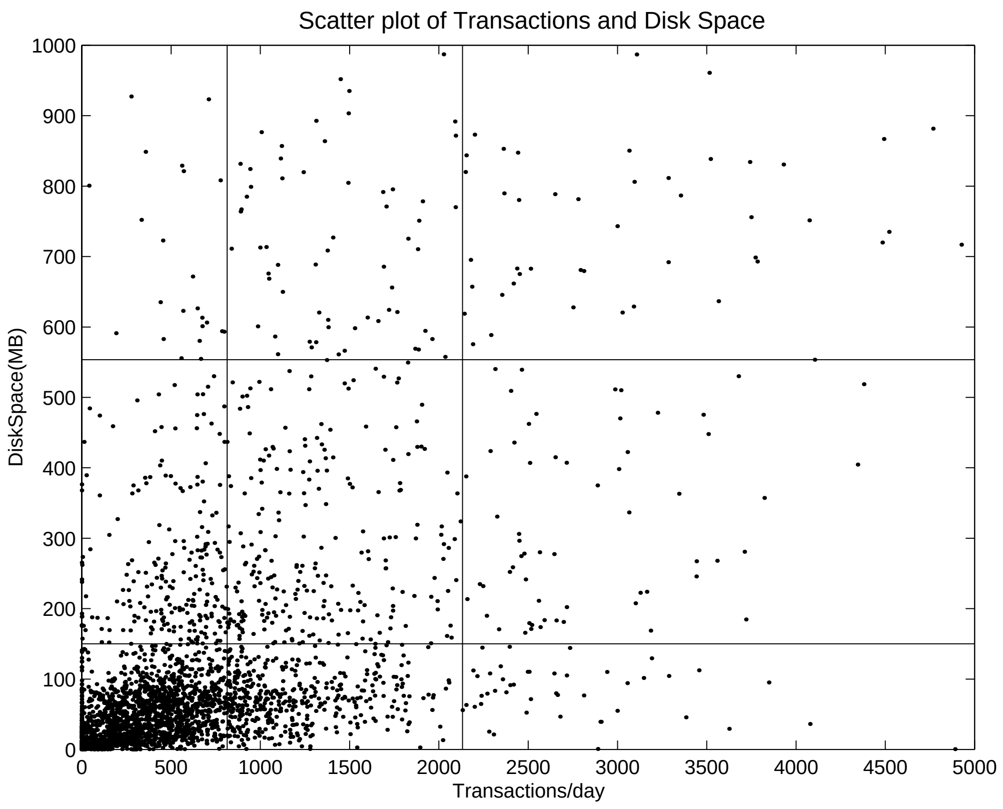

Figure 5.1 The distribution of 8,000 electronic mail databases in activity (Transactions/day) and disk space.

Clearly, most of the databases are located at relatively small values of the resources. Taking the projection of the distribution along the resource dimensions emphasizes this feature of the distribution. Figure 5.2 is a histogram of the transactions/day for the databases of figure 5.1. An exponential distribution function with the same mean is superposed on the data. In practice, the exponential distribution provides a good approximation to the one-dimensional distribution of resource utilization.

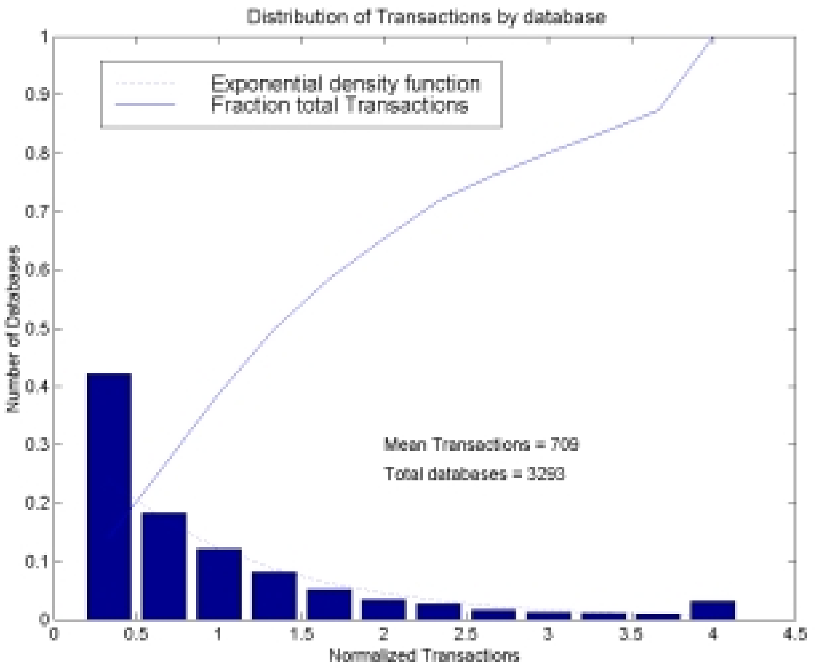

Figure 5.2. A histogram of the activity distribution for the electronic mail databases of figure 5.1. The exponential nature of the histogram is typical of database attributes.

The placement problem is a variation of the multidimensional bin-packing problem in which a set of objects (databases) of given geometry must be placed in containers (servers) so as to evenly fill the containers (Garey and Johnson 1979). System planners may also be interested in the case where one wants to find the minimum number of servers that can support the current databases. Although the general solution is NP hard, the literature shows that heuristic algorithms offer an efficient attack to this problem. There are also mitigating features that simplify the database placement problem, especially for mail databases; 1) Resource consumption for each database is based on historical observations and so contains both statistical and systematic errors. Also, resource consumption is generally growing with time. To allow for measurement uncertainty and growth the total server capacity is usually much greater than the actual amount of resource that must be placed. Thus, the load on each server is acceptable if it falls within a range about the nominal target. 2) The density function is smooth and well approximated by an exponential distribution. Also, servers typically used to host Notes mail in large installations where balancing is important typically support over 1000 mail databases and about a million total RPC's each day. This means that averaging reduces usage fluctuations at the server level. It also simplifies the placement algorithm because anomalies in an individual databases can be compensated by the placement of a group of databases. 3) The bandwidth to the client is independent of which server hosts a database. Bandwidth considerations can be important when the servers are geographically distributed, for example in branch offices. 4) The cost associated with a move compared to the savings achieved by a uniform distribution is not strongly affected by the size of the database.

These considerations, and the success of heuristic algorithms in attacking resource placement problems (ref warehouse location type problems) suggest such an approach be taken here. In particular, a greedy algorithm is used which attempts to equalize the distribution of figure 5.1 across servers by decomposing the problem into regions that are attacked separately. The scatter plot of figure 5.1 shows a simple subdivision of three bins along each dimension leading to nine regions in the plot. The division locations are chosen using the cumulative distribution function for the resource dimensions. Figure 5.2 shows the cumulative resource utilization for an exponential distribution. The high bin on the transaction axis of the figure is chosen to include the databases that account for the top 30% of activity. This is generally the top 10% by number of all databases. The other regions are chosen to include the middle 40% and lower 30% of cumulative transactions. The same subdivision was applied to the disk space axis. Clearly many methods of choosing the regions are possible. In figure 5.3, the sub regions were selected by dividing each transactions region into three areas corresponding to the top 30%, middle 40%, and lower 30% of cumulative disk space utilization for just the databases in each transaction bin. In practice, there was little difference in algorithm performance between choices for the data of figure 5.1. This is a consequence of the distribution of mail databases. The choice of grid in figure 5.3 would probably be a better way to subdivide a distribution that showed strong correlations between activity and transactions and proportionately more databases in the middle and high bins.

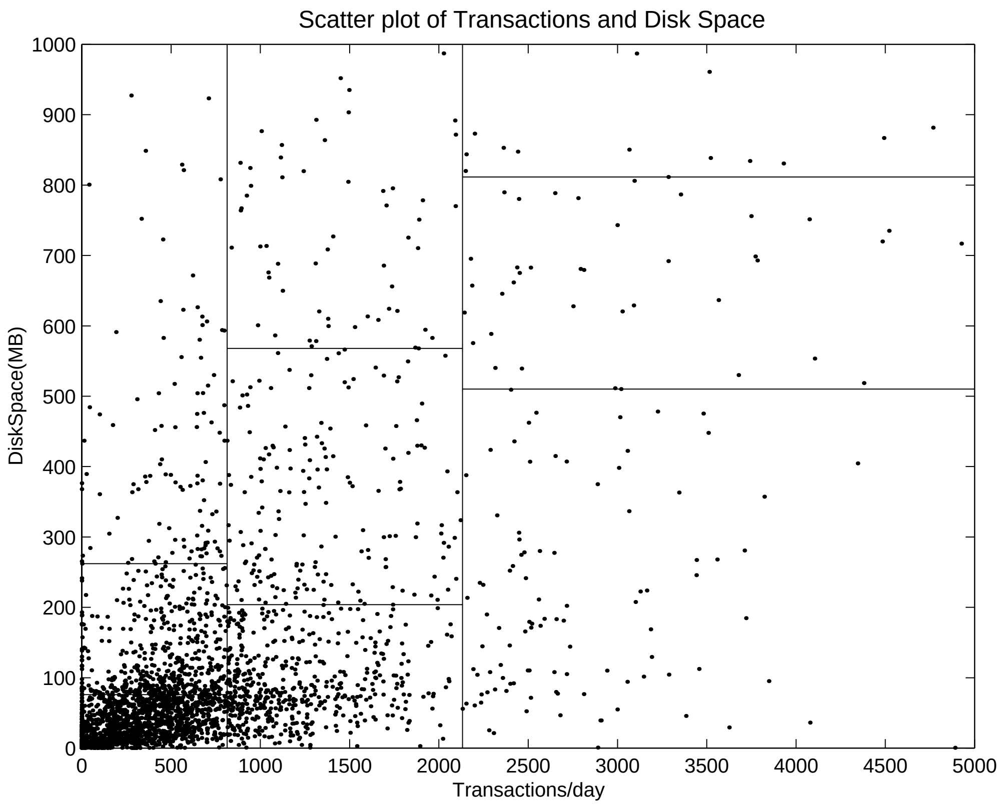

Figure 5.3. The database scatter plot of figure 5.1 has been subdivided into bins for load balancing**.**

Targets for activity and disk space are assigned to each bin. The bin targets are simply the proportion of the server target for that bin. If the server target for transactions is 800,000, and disk space is 1GB, the top 30% transaction bin has a target of 240,000 transactions, and the top disk space bin is 300MB. The algorithm proceeds in two phases. First, a remove phase takes databases from each server until all the bins are below the bin targets in transactions and disk space. The second phase of the algorithm places the databases removed in the first phase on the optimal server. The concept is illustrated in figure 5.4 for two servers S1 and S2.

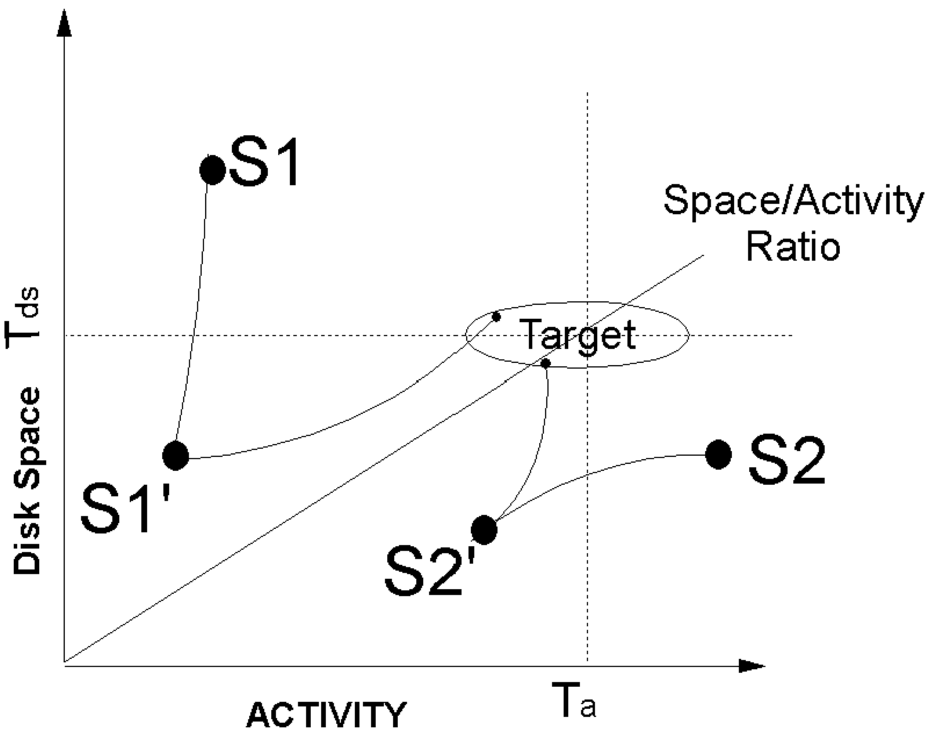

Figure 5.4. The process of balancing servers in multiple resource dimensions. Servers start at the locations S1 and S2 and the algorithm redistributes databases until the servers are within the target region. Because the algorithm has a remove phase followed by a placement phase the trajectory to the target has an intermediate location indicated by the primed labels.

The figure shows the nominal target positions for activity (Ta) and disk space (Ts). The target is indicated as an oval area to reflect the tolerances on the target. The specification of the target tolerance is an input to the algorithm. The default choice is the (RMS) deviation from the target of all servers from the target. The tolerance can also be specified as the maximum acceptable deviation on any of the servers. The activity and space targets (dotted lines) divide the chart into four regions according to whether the server is over or under target for space or activity. A server can be in any of these regions before the balancing algorithm is applied. The figure shows two sample servers at their starting positions S1 and S2. In its starting position, S1 is under target in activity and over in disk space. Databases in each bin are removed from S1 until the server bins are under target in both resources. Because the subsequent placement phase can only move the S1 to higher values along each axis, the initial removal process must arrive at a position S1' which undershoots the final target. The location of S1' is conveniently parameterized as a fraction of the distance from the target values Ts and Ta to the space/activity ratio line for the current position of S1. This fraction is one of the tuning parameters in the algorithm. The closer S1' comes to the space/activity line, the more databases are removed, and the placement phase has more flexibility. In order to move in the most direct path from S1 to S1', the databases in each bin are sorted in ascending order by activity because S1 is further from the activity target than the space target.

The placement phase of the algorithm takes the set of removed databases and attempts to reach the target on each server while achieving the bin proportions dictated by the distribution density.

At this stage, the bins on the server are under target and there is a set of removed databases. This situation suggests an approach based on bin packing. In fact, because of the large number of databases involved a reasonable approach is to divide the problem into a sequence of one-dimensional bin packing problems. In the example of this section, the algorithm alternately attacks the goals of activity or disk space. There are several well known placement strategies for this problem based on sorting the items to be placed in decreasing order and placed according to the criteria of first fit or best fit (Garey and Johnson 1979). For the problem here, the best solution occurred using the criteria of 'worst fit' in which the database is placed on the server having the greatest capacity. This means that the databases are sorted in descending order for the current resource goal and the algorithm places the databases sequentially from the list. The transactions and disk space for each database determine bin membership. Only servers with room in both dimensions of the bin are candidates to receive the database. The database is placed on the server in the candidate set with the bin having the most room under target in the current resource dimension. For example, suppose the database has 4,000 transactions/day and 700MB of disk space. According to figure 5.1 it is in the high bin for both transactions and disk space. If the algorithm goal is activity and there are two candidate target servers, the database is placed on the server with the most room in activity. The situation is shown in figure 5.5. Although the database fits in both bins, it would be placed on server 'B' because 'B' has the largest capacity in activity. When the goal switches to disk space, the database of the figure would be placed on server 'A'.

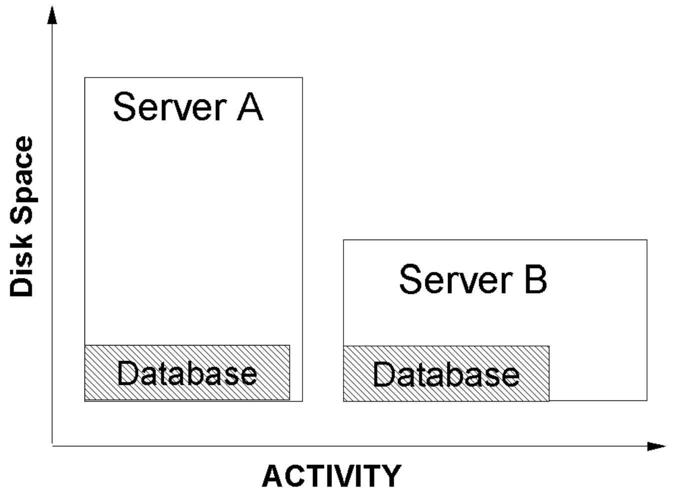

Figure 5.5. Placement decisions for databases depend on the a maximum capacity criteria for the activity and disk space goals.

The placement phase switches between the activity and disk space goals. The algorithm can be configured to switch using several criteria. In the method of steepest descent, the algorithm works on the resource that is furthest from the target. The distance from the target is updated after every N database placements. N is an algorithm tuning parameter, typically a few percent of the initial length of the remove list. When both resources are within target tolerance, databases on the remove list are sent back to their originating server to minimize the number of database moves. In the round robin approach, the goal is switched every N database placements independent of the distance from the target. Both methods work reasonably well, sometimes one better than the other depending on the initial distribution of databases. The ability of the algorithm to achieve its goals is often dependent on the initial distribution of databases. A difficult starting configuration can cause the algorithm to miss its goals. For this reason, the implementation allows iterative execution. A second run often results in a better solution, moving a small fraction of the databases moved on the primary execution of the algorithm.

The results of applying the algorithm are indicated in figures 5.6 and 5.7. Figure 5.6 summarizes the initial distribution of databases in a set of servers before the algorithm is applied. Each stacked bar summarizes the distribution of activity on a server. The three components of a bar indicate the amount of activity in the low, middle, and high bins for that server. Figure 5.6a shows the activity distribution, while figure 5.6b is the space distribution. The server distributions after the algorithm is applied are indicated in figure 5.7. In the transition from figure 5.6 to figure 5.7, 30 percent of the databases were moved.

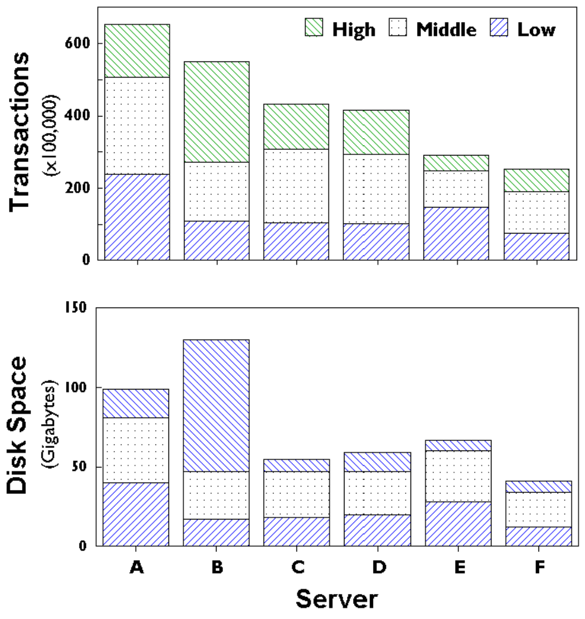

Figure 5.6. Distribution of databases in activity and disk space on six mail servers prior to execution of the balancing algorithm.

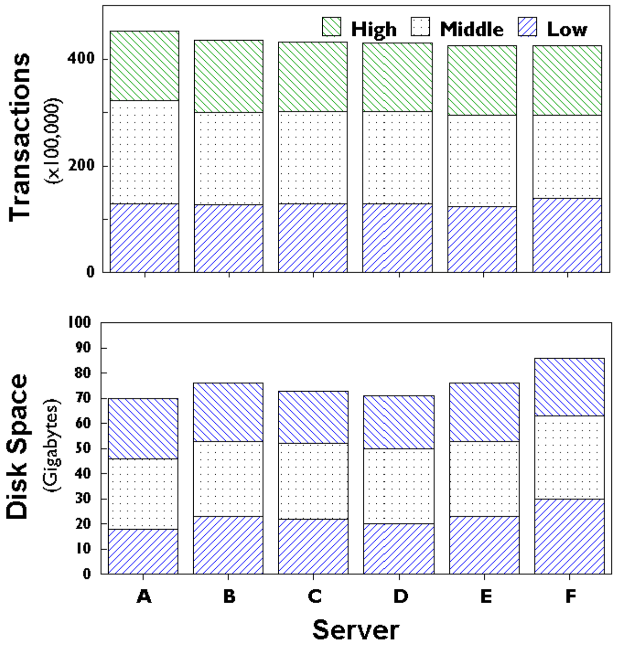

Figure 5.7. Distribution of databases in activity and disk space subsequent to execution of the balancing algorithm. In addition to meeting the target, the profiles of the distributions are approximately equal.

The coefficients of variation (CV) after the load balance for the activity and space targets are 0.02 and 0.07. These values correspond to the data of figure 5.7. The following table shows that CVs in other resources were also significantly reduced even though they were not explicit targets in the algorithm.

Results non-target Resources

Resource	*Before*	*After*
Read/Write ratio	0.6	0.2
Bytes From	0.35	0.1
Bytes To	0.8	0.2

Finally, we address the performance and scalability of the algorithm. The initial phase of load balancing in which databases are removed until the servers are under goals is greedy. This requires sorting the databases in the server bins by activity or disk space. Therefore we expect the execution time to have terms of O (n) and O (n*log (n)) (using a merge sort) where n is the average number of databases on each server. In the subsequent placement phase, each database is placed on the best server using a simple scan of the M candidate servers. So performance will be O (n*M). The following table shows the results of running the algorithm on a 300MHz Pentium III processor with 256MB of memory. The performance is dominated by the linear terms up to 13 servers with a total of 7700 databases.

Algorithm Performance

# Servers	*Total DBs*	*Moved DBs*	*Time (s)*
3	2630	795	6
7	4708	1494	8

10	6114	1805	10
13	7756	2317	13

Acknowledgements.

We benefited from discussions about these concepts with D.H.J.Epema and thank Joe Hellerstein for his insights on the application of Little's Law. We thank Lorrie Renz for assistance with data collection from production Domino servers within IBM.

References

Mark Crovella, Mor Harchol-Balter, and Cristina Murta, 1997, "Task Assignment in a Distributed System: Improving Performance by Unbalancing Load," *Proceedings of ACM Sigmetrics '98 Conference on Measurement and Modeling of Computer Systems* June 1998, Madison, WI

Dan, A. and Sitaram, D. 1995, "An Online Video Placement Policy Based on Bandwidth to Space Ratio (BSR)", In SIGMOD'95 Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.

Demers, A., et al, 1988, "Epidemic Algorithms for Replicated Database Maintenance", ACM Operating Systems Review, January, 1988

Epema, D. H. J., 1999, "Load Balancing in Replicated Databases with Loose Consistency Requirements", 10. GI/ITG-Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen (MMB'99), Forschungsbericht Nr. 99-17, Trier, Germany, pp. 31-35, 22-24 September 1999.

Garey, Michael R., and Johnson, Davis S., 1979 "Computers and Intractability", W. H. Freeman and Co.

Goldszmidt, G, and Hunt, G. 1999, "Scaling Internet Services by Dynamic Allocation of Connections", Proceedings of the sixth IFIP/IEEE international symposium on Integrated Network Management, Boston, MA.

Hellerstein, J. 1999, private communication.

Hellerstein, Zhang, and Shahbuddin 1999, "An Approach to Predictive Detection for Service Management ", Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated NetWork Management, Boston, MA. May 1999

Kleinrock, L., 1975, "Queuing Systems Volume 1:Theory", Wiley-Interscience, ISBN 047491101.

Kremien, K, and Kramer, J. 1992, "Methodical Analysis of Adaptive Load Sharing Algorithms", IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 6, November 1992

Lamb, J., and Lew, P. 1999, "Lotus Notes & Domino 5 Scalable Network Design", McGraw-Hill, ISBN 007913792X

Malone, T.W., Grant, K.R., Lai, K., Rao, R., Rosenblitt, D., 1987 " Semi Structured Messages Are Surprisingly Useful for Computer Supported Coordination", ACM Transactions on Office Information Systems, Vol. 5, pp 115-131, April, 1987

Mahmoud, S. A., and Riordon, J.P. 1976 "Optimal Allocation of Resources in Distributed information Networks", ACM Transactions on Database Systems, Vol 1, pp 66-78, 1976

Mitzenmacher, M. 1996, "The Power of Two Choices in Randomized Load Balancing", Ph.D. thesis, University of California, Berkeley, CA.

Pacitti, E, and Simon, E., 2000, "Update Propagation Strategies to Improve Freshness in Lazy Master Replicated Databases", Very Large Database Journal (VLDB), 8, pp 305-318

Pope, W. G. 1998, "Characterizing Lotus Notes clients", IBM Research Report RC21252

Pope, W.G., and Mummert, L. 2000, "Using capacity space methodology for balancing server utilization: Description and case studies", IBM Research Report RC 21828

Terry, D.B, Theimer, M.M., Petersen, K., Demers, A.J., Spreitzer, M.J., and Hauser, C.H. 1995, Proceedings ACM Special Interest Group on Operating Systems (SigOps) December 1995

Tetzlaff, W.H., 1979, "State Sampling of Interactive VM/370 Users", IBM Systems Journal Vol. 18, No. 1, pp. 164-180.

Wolski, R., Spring, N., Hayes, J. 1999, "Predicting the CPU Availability of Time-shared UNIX Systems on the Computational Grid", Proceedings of the 8th High Performance Parallel and Distributed Computing Conference (HPDC8), August 1999. (Also available as UCSD Technical Report Number CS98-602)