Effects of User Request Patterns on a Multimedia Delivery System*

Christopher B. Mayer K. Selçuk Candan
Venkatesh Sangam

Computer Science and Engineering Department
Arizona State University
e-mail: {chris.mayer,candan,venkatesh.sangam}@asu.edu

Abstract

We recently introduced a novel method for creating replication systems where the replicated objects' sizes and/or per-object service times are large [10]. Such replication systems are well-suited to delivering multimedia objects on the Internet. Assuming that user request patterns to the system are known, the method creates replication systems that distribute read load fairly to servers so that the likelihood of servers overloading is reduced. Thus, the systems produced are highly available and responsive to user requests. In this paper, we report on results that reveal (*i*) how server loads are affected and (*ii*) the impact of two system design parameters (indicators of a system's load distribution qualities) have on server load when user request patterns differ from that for which a system was designed.

1 Introduction

Replication is an accepted method for improving availability and response times of Internet content. The main idea behind replication is that storing copies of an object (file, database, web page, etc.) on servers throughout a network provides single points of failure. Since the object is available at many servers, high demand loads can be met and the failure of an individual server does not make the object inaccessible.

Multimedia content can be delivered over the Internet and can benefit from replication. In this paper, we focus specifically on systems for delivering (making available for download) multimedia content. When designing a multimedia delivery system the following considerations and constraints apply.

Read Load: In a multimedia delivery system, an object is written by its author and remains accessible for some period of time before it is removed. Since an object is supposed to be a finished product, it is rarely updated once in publication. Therefore, the load on a system's servers is due mainly to handling users' read requests and the load for writing can be ignored. Since we are concerned with only the read request load (read load), we will use the term "load" and "read load" interchangeably.

Content Size: Multimedia objects such as video files tend to be large (tens or hundreds of megabytes) and therefore require special consideration when being replicated.

- Large objects can not be rapidly replicated in response to fluctuating demand. Therefore, it is sensible to pre-position multimedia objects.
- While pre-positioning is good, over-provisioning can be bad. Creating too many copies of an object having relatively low demand is wasteful. Replication costs for an object should be relative to demand for the object.

Service Times: Even if users connect to a server over broadband connections, delivering a multimedia object to a user requires the server's attention for a long period of time. These **long service times** occur because the content is either large or streamed, or both.

Server Behavior: A common behavior of servers is that they can support multiple simultaneous requests while maintaining acceptable quality of service. Once a server's load capacity is exceeded, the server's service quality rapidly declines, resulting in stalled requests and disappointed users. In combination with the service time property above, this behavior suggests that the best way to maintain the availability of a multimedia object and keep users happy is to ensure that servers operate below their load capacities.

*This research funded by NSF grant 998404-0010819000.

Techniques for easing the load on, or improve the performance of, multimedia servers (mainly video and streamed media servers) include: caching [1, 6, 11, 14, 15], protocols for stream and download sharing [5, 8, 11], and customized server designs [4, 5, 7]. While all these approaches are beneficial, they all depend on access to the content's source. Thus, they are not a panacea for availability and responsiveness; abundant access to the source content, as replication provides, is required.

The profusion of special techniques for video and streamed media delivery are indirect signs that multimedia should be stored and delivered separately from other types of Internet content. Towards this end, several video-only delivery schemes have been proposed ([3, 12, 13] for instance). Common weaknesses of these schemes are that content is assumed to come from a single source (i.e., only one entity is using the system) and the delivery network is a tree (which networks are commonly not).

In a previous work [10], we introduced a replication architecture and an accompanying design method well-suited for delivery of multimedia content, or any other Internet content where service times or object sizes are large. In our approach, servers are organized into write-sets and read-sets and requests are fulfilled using a specific read protocol. Our system structure is more general than that of the tree-based video distribution systems and freely handles multiple content providers. In our system, a server's share of system load is proportional to the server's contribution to total system read load capacity. In other words, load is distributed *fairly* to the servers. Fairly distributing load minimizes the odds that a server will exceed its load capacity. Since servers operate under their load limit, the system is responsive and content is highly available.

In [10], we rigorously examined the implications of our server organization and read protocol and showed that load fairness is a complex non-linear condition that can not be solved easily. Therefore, in [10] we derived two key parameters: σ (describing good request distribution strategies) and N (describing good server organization) that reflect a system's load fairness. Using σ and N, we developed and tested a design method for quickly designing replication systems with nearly optimal load fairness.

In this paper, we further the work begun in [10] with a study of (*i*) server loads and (*ii*) the importance of σ and N to server loads when the pattern of read requests entering a system differs from that for which the system was designed. In Section 2 we review our replication approach. We then we explain why divergent request patterns require further investigation in Section 3. Next, we describe how we conducted experiments to evaluate the impact of σ and N (Section 4). Section 5 contains the results of our study and Section 6 summarizes.

2 A Replication System Suitable for Multimedia Content Delivery

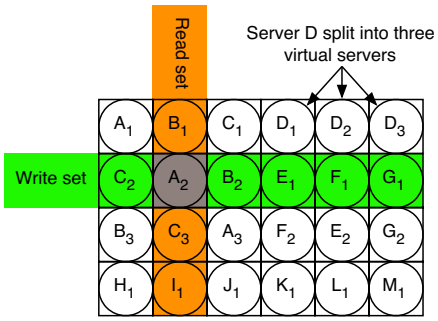
In this section we describe the features and mechanics of our method for creating a replication system. A detailed accounting of the material summarized here can be found in [10].

2.1 System Structure

A replication system has a set of servers, $S = \{s_1, s_2, \ldots, s_n\}$, which it uses to replicate objects. In our approach, we organize these servers into intersecting subsets called **write-sets** and **read-sets**. To write an object, a write-set is chosen and the object is replicated on each server in the write-set. To read an object, a read-set is chosen and the object is delivered to the requesting user from a server in the selected read-set. It is allowable for an object to be written to multiple write-sets. We ensure that every written object can be accessed from any read-set by requiring that each read-set and each write-set have at least one server in common.

Although there are many ways to construct write-sets and read-sets while maintaining this requirement, we limit ourselves, for simplicity reasons, to the special case where the write-sets and read-sets are determined by arranging servers in a grid. In a grid fully populated with one server per grid cell, rows correspond to write-sets and columns to read-sets. As Fig. 1a indicates, each read-set intersects every write-set with at least one server. Some readers may notice that this grid-based structure resembles grid-based quorum systems ([2] and [9] for example). This is intentional since quorum systems feature decentralized operation and have the potential load-balanced server operation.

[1] Due to the unpredictable nature of Internet routing and to generalize for any request routing scheme, we do not specify how requests are



(a)

Read Protocol

1. A user generates a request for an object (an **initial-request**) which is directed to one of the system's servers (a **proxy**). The distribution of a particular user's requests to the proxies is called the **user request pattern**. [1]
2. The proxy selects a read-set using a preset, probabilistic **proxy strategy**.
3. The proxy identifies the server(s) with the most up-to-date copy (in case the object has been updated) of the object in this read-set. If more than one server has the most up-to-date object, one of the servers is picked equiprobably to serve the data.
4. The proxy redirects the user to the server it has selected.
5. The user receives the object from the appropriate server, thus inducing a read load on the server.

(b)

Figure 1: A grid-based replication system where rows are write-sets and columns are read-sets is shown in (a). The read protocol for the replication system is shown in (b).

2.2 Read Protocol

An important part of our replication system is the read protocol shown in 1b. Note that locating the appropriate server within a read-set adds to the delay in responding to requests. However, this delay is extremely small compared to the time required to serve a multimedia object and is unlikely to detract much from the user's experience. Further, this delay can be minimized by using a directory service, especially since object locations rarely change.

Based on the structure of write-sets and read-sets and the read protocol, we now the issue of load fairness.

2.3 Fairness

To minimize the likelihood of a server overloading, a server should experience a load proportional to its contribution to the system's total capacity. In other words, system load should be distributed *fairly* to servers. This will ensure that no server is pushed beyond its capacity unless the whole system is. This can be accomplished by

- identifying a good assignment of servers to write-/read-sets (thus defining the system's structure) and
- selecting effective proxy strategies for directing initial-requests to read-sets.

As a first step towards fairness, we further refine the system's structure by splitting each server into one or more **virtual servers**. Each server in the system, $s_i \in S = \{s_1, s_2, \ldots, s_n\}$, is represented by its read load capacity, $L_{max,i}$. Denoting a base capacity as L_{base}, we split each server $s_i \in S$ into $n_i = \lfloor \frac{L_{max,i}}{L_{base}} \rfloor$ virtual servers, $V(s_i) = \{s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}\}$. Populating the cells of a grid with virtual servers (Fig. 1), instead of regular servers, results in nearly equal amounts of capacity at each grid cell. If the system read load can be distributed equally to each grid cell, then we have achieved the goal of fairness. Unfortunately, the load on the virtual server in each grid cell, and in turn on the servers, depends on (*i*) the arrangement of virtual servers in the grid (the **grid mapping**), (*ii*) the policy for deciding to which write-sets an item should be written (**write-policy**), (*iii*) initial-request loads at the proxies determined by the user strategies, and (*iv*) the proxy strategies. Hence, load fairness requires more than placing virtual servers in the grid.

Given a grid of l write-sets (rows) and k read-sets (columns) populated with virtual servers, a total system read load, L, the read load of server s_i, $L(s_i)$, is:

$$L(s_i) = L \cdot \sum_{rs=1}^{k} \sum_{ws=1}^{l} pr(ws) \cdot pr(rs|ws) \cdot pr(s_i|rs \wedge ws) \quad (1)$$

In the above equation, $pr(ws)$ denotes the probability that write-set ws contains a server with the requested content; $pr(rs|ws)$ denotes the probability that read-set rs is chosen by a proxy given that write-set ws contains the content;

directed to servers. Instead, we rely on the observation that request routing is somewhat predictable and can be expressed probabilistically.

and $pr(s_i|rs \wedge ws)$ denotes the probability that server s_i is selected for serving the request given that write-set ws contains the requested content and read-set rs is chosen.

To obtain load fairness, we need to ensure that the grid mapping and proxy strategies distribute the total system read load onto individual servers in proportion to each server's contribution to total system capacity:

$$L(s_i) \approx L \cdot \frac{L_{max,i}}{\sum_{s_j \in S} L_{max,j}} = \frac{L \cdot L_{max,i}}{l \cdot k \cdot L_{base}} = \frac{L \cdot n_i}{l \cdot k}. \tag{2}$$

Note that this **fairness condition** is a complex non-linear equation, and solving it directly is expensive. Also, it is not straightforward to find a mapping and determine proxy strategies using (1) and (2) directly. Therefore, we use (1) and (2) to identify parameters, σ and N, that can be used to construct highly fair replication systems.

2.4 Deriving σ and N

To begin the derivation of σ and N, we assume that the write-policy *distributes objects to write-sets such that the request load for each write-set is equal*. That is, $\frac{1}{l}$ of the system load is directed to the virtual servers in each write-set. With this assumption (1) becomes

$$L(s_i) = \frac{L}{l} \cdot \sum_{rs=1}^{k} \left(pr(rs) \cdot \sum_{ws=1}^{l} pr(s_i|rs \wedge ws) \right). \tag{3}$$

Using this equation, we can rewrite the fairness condition, (2), as

$$\frac{n_i}{k} = \sum_{rs=1}^{k} \left(pr(rs) \cdot \sum_{ws=1}^{l} pr(s_i|rs \wedge ws) \right). \tag{4}$$

Note that, if a request can be served from write-set ws and a proxy chooses read-set rs, then one of the servers having a virtual server in the intersection of ws and rs will be selected to serve the content. If there is more than one server in the intersection, then each server has an equal chance of being chosen. Consequently, if we let

- S_{ws} be the set of servers that have virtual servers in write-set, ws; i.e., $S_{ws} = \{s_i | v \in V(s_i) \wedge v \in ws\}$;
- S_{rs} be the set of servers that have virtual servers in read-set, rs; i.e., $S_{rs} = \{s_i | v \in V(s_i) \wedge v \in rs\}$; and
- $V_{ws,rs}$ be the set of virtual servers in read-set rs that have a corresponding server in write-set ws; i.e., $V_{ws,rs} = \{v \mid v \in V(s_i) \wedge s_i \in S_{rs} \wedge s_i \in S_{ws}\}$,

then, $pr(s_i|rs \wedge ws) = \frac{|V(s_i) \cap V_{ws,rs}|}{|V_{ws,rs}|}$. Hence, (4) becomes

$$\frac{n_i}{k} = \sum_{rs=1}^{k} \left(pr(rs) \cdot \sum_{1 \le ws \le l} \frac{|V(s_i) \cap V_{ws,rs}|}{|V_{ws,rs}|} \right). \tag{5}$$

Notice that (5) has two terms that can be manipulated: $pr(rs)$ and $\frac{|V(s_i) \cap V_{ws,rs}|}{|V_{ws,rs}|}$. The first term is a function of the proxy strategies and the latter depends on the grid mapping. By isolating these two terms, we gain a measure of insight into how to construct an optimally fair system.

2.4.1 Isolating Proxy Strategies: Parameter σ

In order to extract the term related to proxy strategies, we isolate the term in (5) related to read-set selection by proxies to get

$$\forall_{1 \le rs \le k},\ pr(rs) = \frac{1}{k} \tag{6}$$

which says that the fraction of requests directed to read-set rs should be inversely proportional to the number of read-sets, k. In other words, initial requests should be directed in equal amounts to each column. This implies that

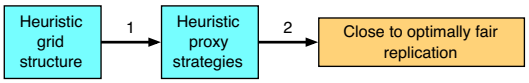


Figure 2: Replication using σ and N

the combined effect of all the proxy strategies should ensure an equal distribution of initial request to each column. We denote the fraction of initial read requests directed to read-set rs as $\sigma(rs)$, or **read-set σ-value**. The ideal σ-value for a read-set is $\frac{1}{k}$. If $\sigma(rs)$ is greater (less) than the ideal σ-value, then rs, and the virtual servers in rs, will receive more (less) than their fair share of system load. Likewise, since a server's load is the sum of the load on its virtual servers, the higher the σs of the read-sets in which a server has a virtual server (**average server σ**), the more load the server will receive.

2.4.2 Isolating the Grid Mapping: Parameter N

Assuming that all read-sets have the ideal σ-value of $\frac{1}{k}$, we can reduce (5) to

$$\forall s_i \in S, \sum_{rs=1}^{k} \sum_{ws=1}^{l} \frac{|V(s_i) \cap V_{ws,rs}|}{|V_{ws,rs}|} = n_i. \tag{7}$$

This equation can be satisfied by ensuring that

$$\forall_{s_i \in S} \forall_{1 \le rs \le k} \forall_{1 \le ws \le l} \frac{|V(s_i) \cap V_{ws,rs}|}{|V_{ws,rs}|} = \frac{n_i}{k \cdot l} \tag{8}$$

holds. The left-hand side of (7) captures the degree of content overlap of server s_i with other servers that share both read-sets and write-sets with s_i. Isolating this overlap we get

$$N(s_i) = \sum_{rs=1}^{k} \sum_{ws=1}^{l} \frac{|V(s_i) \cap V_{ws,rs}|}{|V_{ws,rs}|}. \tag{9}$$

$N(s_i)$, or **server N-value**, indicates s_i's vulnerability to being selected for serving a read request. The ideal value for $N(s_i)$ is n_i, the number of virtual servers s_i has. If $N(s_i)$ is too high ($> n_i$) or too low ($< n_i$), then s_i will be selected too often or not often enough and will not receive its fair share of load.

As an example of how to calculate an N-value, consider server A and the second write-set and second read-set highlighted in Fig. 1. Server A has three virtual servers, thus $V(A) = \{A_1, A_2, A_3\}$ and $n_A = 3$. Three virtual servers in the second read-set have a server in the second write-set, so $V_{2,2} = \{A_2, B_1, C_3\}$ (i.e., if read-set 2 is chosen by a proxy and the requested content is on a server in write-set 2, then three servers can be selected for download). Since $V(A)$ and $V_{2,2}$ only have A_2 in common. $|V(A) \cap V_{2,2}| = 1$. Thus, $\frac{|V(A) \cap V_{2,2}|}{|V_{2,2}|} = \frac{1}{3}$ (i.e., given that a request is for content contained in write-set 2 and read-set 2 was chosen, server A has a 1-in-3 chance of serving the request). Repeating these calculations for server A for all rows and columns and summing the results shows that A has a perfect N-value: $N(A) = n_A = 3$.

2.5 Replication Based on σ and N

To produce the fairest system possible, σ- and N- values must be as close to their ideal values as possible. While σ and N indicate how fair a system is, they are derivatives of the complex, non-linear fairness condition and, therefore, can not be used directly to construct a replication system either. However, they do provide insight as to how a fair replication system should look. In [10], we exploited σ and N to develop a two-step heuristic approach for creating a replication system (Fig. 2). In the first step, we set the system's structure by mapping virtual servers to grid cells so that each server has good N-values. This structure is then used to formulate proxy strategies that result in the best possible σ-values for each grid column.

Given a set of servers and a grid,

1. Put the servers into groups such that servers in each group have the same number of virtual servers.
2. Try to fill the grid with the given clusters
3. If such a filling is not possible, break some of them into smaller clusters to fit them into the grid.

The result is a server-to-grid mapping.

(a)

Given a server set, a grid, and server-to-grid mapping, and user strategies,

1. Identify the linear constraints for
   (a) σ-optimality,
   (b) write-policy dependence
   (c) fairness, and
   (d) column selection restrictions on the proxies.
2. Solve while minimizing error terms in the constraints.
3. Extract proxy strategies from the solution.

The proxy strategies result in optimal σ-values.

(b)

Figure 3: Pseudocode for (a) the cluster-based mapping algorithm and (b) producing σ-optimizing proxy strategies.

Pseudocode for mapping a grid is shown in Fig. 3a. The algorithm clusters virtual servers and then maps the clusters to the grid. Clustering limits the interaction between servers that gives imperfect N-values. If all virtual servers can be mapped, while maintaining cluster integrity, then all servers will have perfect N-values. Note that clusters must sometimes be split into smaller pieces in order to facilitate placement. This can create imperfections in N-values. However, as we showed in [10], the negative effects of splitting are minimal.

Once the system's structure is set, we use the structure to formulate a linear program (LP) and solve the LP to find proxy strategies for the system. In addition to the system's structure, the LP considers other fairness-related factors such as the system's write-policy and user request patterns. Figure 3b shows pseudocode for the LP's construction and extraction of proxy strategies. The specifics of the LP can be found in [10]. The output of the LP are the proxy strategies that produce the best possible (closest to ideal) σ-values for each read-set. For each proxy, its particular proxy strategy gives the frequency at which it should select a read-set when handling initial-requests from users.

In [10] we studied the impact of σ and N on server load fairness. We showed that grid-based replication systems constructed using our two-step approach are highly fair when operating conditions are exactly that for which the system was designed. In this paper, we investigate what happens when user request patterns no longer match the patterns for which the system was designed. Specifically, we examine (*i*) the importance of σ and N to load fairness and (*ii*) how server loads are affected user request patterns deviate from expectations.

3 Divergent Initial-Request Loads

Since proxies redirect client initial-requests to servers, the performance of the replication system depends on the expected distribution of the users' initial-requests to proxies. As explained earlier as part of the read protocol, the probability that a given user's initial-requests arrive at a certain proxy, is given as a distribution function called the **user request pattern**. User request patterns (or at least their cumulative effect on the proxies) are an input to the linear program (LP) that is solved to get the proxy strategies for selecting read-sets. Thus, a replication system is tailored for the user request patterns input into the LP. Since user request patterns are will change over time, using a fixed user request pattern to construct a replication system is a potential weakness of our approach. If request patterns change too much, server load fairness could be lost and the system would perform poorly.

In the remainder of this paper, we present two kinds of results, obtained experimentally, about our proposed replication system.

- We show the importance of σ and N to load distribution when user request patterns deviate from those for which the system was designed.
- We show that systems built using our two-step construction approach (see previous section) are resilient to changes in user request loads.

4 Experimental Setup

In order to test the performance of replication systems that use our write-/read-set structure and read protocol, we have constructed a testbed system that uses real web servers. The use of real servers adds a degree of realism that

ordinary simulation does not provide. Because of space constraints we can not go into the details of the testbed system in this paper. However, we do describe the conditions for conducting experiments.

To prepare for an experiment, servers are arranged into their write-sets and read-sets using the grid structure and given their proxy strategies (calculated in advance based on expected loads). A different object is written to each write-set (a row of the grid). All data items have the same size and hence the same download times. Having each object be the same size and having each write-set contain a single object captures the effects of a perfectly tuned write-policy. The download time of an object is simulated by having servers execute a sleep operation of 5 seconds. The running time for an experiment is 40 times the sleep time, or 200 seconds. This is the minimum time needed for an experiment to show long-term loading behavior. Once the setup stage is complete, the experiment can begin.

Performing an experiment consists of generating user requests for objects and the handling of those requests by the system. User requests are regularly-spaced over a second to meet a specified request rate. For example, if the request rate is 10 requests/sec, then a new request is generated every tenth of a second. Uniform request generation, while simple, is adequate since object sizes, and hence download times, are relatively large compared to request inter-arrival times. For each request, an object is selected uniformly at random. The proxy server that will receive a newly generated initial-request is selected at random using a probability distribution that models the effects of the user request patterns.

Experiments were performed using 20 sets of servers. A server set is the servers available for use by a replication system. Server sets were generated so that the number of virtual servers in each set equalled 64 and would fill an 8x8 grid. The number of virtual servers per server was randomly generated according to the following distribution: 40% of the servers have 1, 30% have 2, 10% have 3, 10% have 4, and 10% have 5 virtual servers.

In order to observe the effect of N as request patterns change, we map a server set to a grid using two different mapping strategies:

- **Random.** This strategy randomly maps virtual servers to a grid. This results in server N-values that differ greatly, both up and down, from their ideals.
- **Cluster.** Grids are mapped using an algorithm based on the clustering pseudocode of Fig. 3a. Clustering results in ideal or nearly ideal N-values for all servers in a grid.

To observe the effect of σ, we used two methods for formulating proxy-strategies that result in favorable and unfavorable σ-values.

- **Not σ-optimized.** Each proxy redirects initial-requests equiprobably to the read-sets (grid columns) in which it has virtual servers. As such, read-set σs can vary greatly, being highly influenced by the system's structure.
- **σ-optimized.** Here a linear program is formulated and solved to obtain proxy-strategies that produce optimal σs. As with the above non-σ-optimized strategy, proxies can only redirect initial-requests to read-sets in which the proxy has virtual servers. Even with this restriction, resulting σs are close to ideal regardless of mapping strategy.

Mixing mapping and proxy strategies results in four replication systems (mapping/proxy systems or **MP-systems**) for each server set. The mix of good and bad σ- and N- values in the four systems allows us to observe the influence of σ and N on server load as initial-request loads at the proxies vary in response to changing user request patterns. We refer to an MP-system by the mapping method used and the presence of σ-optimization as shown in Fig. 4 and listed below.

RANDOM: not N- or σ-optimized	RANDOM-σ: not N-optimized, but σ-optimized
CLUSTER: N-optimized, but not σ-optimized	CLUSTER-σ: N- and σ-optimized.

For the tests, a **baseline load** of 6.4 requests per second is the arrival rate of initial-requests to each proxy (256 requests per proxy divided by the 40 second experiment length). Thus, the σ-optimized systems were constructed for user request patterns whose cumulative effect is that each proxy is equally loaded with initial-requests.

In order to systematically explore the effects of varying user request patterns, we randomly selected subsets of proxies in each server set and subjected them to increased initial-request loads. [2] We refer to the combination

[2] Note that we only increase initial-request loads at proxies. Since Internet demand only grows over time and unevenly, this is a reasonable thing to do.

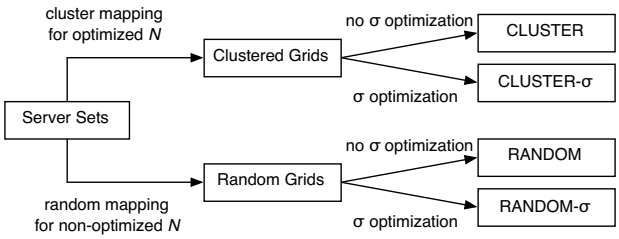


Figure 4: Naming concept for the four types of replication systems created from a server set.

of proxies selected to receive extra load and the extra load they are given as an **initial-request-combination** (**IR-combo**). For a given server set, there are seven IR-combos which form an **initial-request-set** (**IR-set**). An IR-set is built as follows. The first IR-combo in the set is each proxy receiving the baseline load In this combination, 0% of the proxies receive 0% extra load. We call this the 0%-0% IR-combo or the **baseline system**. Next, three proxy subsets of sizes $\lceil 10\% \rceil$, $\lceil 20\% \rceil$, and $\lceil 40\% \rceil$ of the number of proxies in the server set are formed, with the larger subsets reusing servers from the smaller ones. The proxies in these subsets will receive 25% and then 50% extra initial-request load above the baseline load. Combining the proxy subset sizes and extra request percentages produces the remaining six IR-combos in an IR-set: 10%-25%, 10%-50%, 20%-25%, 20%-50%, 40%-25%, and 40%-50%.

Example 4.1 We now illustrate how to create an initial-request-set. Consider a server set with thirty servers numbered 1 through 30 and a baseline initial-request load of 10 requests per second (req/sec).

1. The 0%-0% combination is all proxies receiving 10 req/sec.
2. For the 10% proxy subset we pick three proxies, say 5, 9, and 21. For the 10%-25% combination, we increase the number of requests to these three proxies by 25%; they will each receive 12.5 req/sec. Proxies not in the subset still get only 10 req/sec. To create the 10%-50% combination, initial-requests are increased by 50% to 15 req/sec at the three proxies.
3. To build the 20% subset, $\{2, 5, 9, 11, 21, 29\}$, the 10% subset is augmented by three more servers: 2, 11, and 29. Combinations 20%-25% and 20%-50% are created by increasing the initial loads at these proxies by 25% and 50%, respectively.
4. The 40% subset is the 20% subset plus servers 1, 15, 17, 23, 24, and 28. Increasing the initial-requests by 25% and 50% at the selected proxies gives combinations 40%-25% and 40%-50%. ◇

By running a server set through each of its four MP-systems and each of its seven IP-combos (each server set is run 28 times), we can detect trends in server read loads and compare the influence of N and σ on server load as user request patterns vary.

5 Results and Observations

In this section, we answer six questions about the effects of σ and N when initial-request loads to proxies diverge from their expected values. To do this, we observe the **extra load** experienced by a server when operating as part of the four MP-systems created from the server set of which the server is a member. Extra load is the difference in a server's read load in an experiment where user request patterns have changed and in an experiment where user request patterns are exactly what the system was designed for (the 0%-0% IR-combo or baseline system). Server read load is the average number of reads (downloads) experienced by a server in each second of an experiment. For example, if server s had a load of 5 read requests per second in a baseline system experiment and then had a load of 7 requests per second in a 20%-50% combination then s's extra load is 2, an increase of 40%.

The evidence supporting answers to Questions One through Four involve twenty different server sets. Questions Five and Six are answered using results from four rounds of repeated experiments on the second of the twenty server sets. In the four rounds, the initial-request-sets were not changed. Since space is limited, we present evidence representing behavioral trends seen throughout all the experiments.

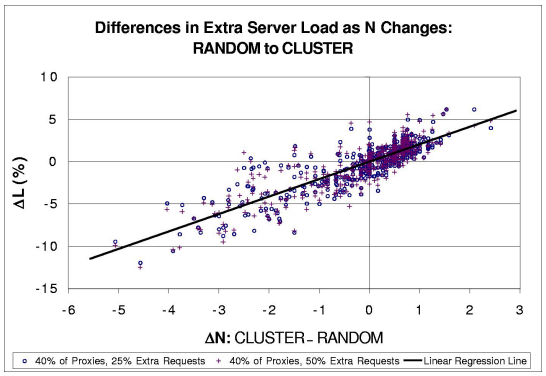
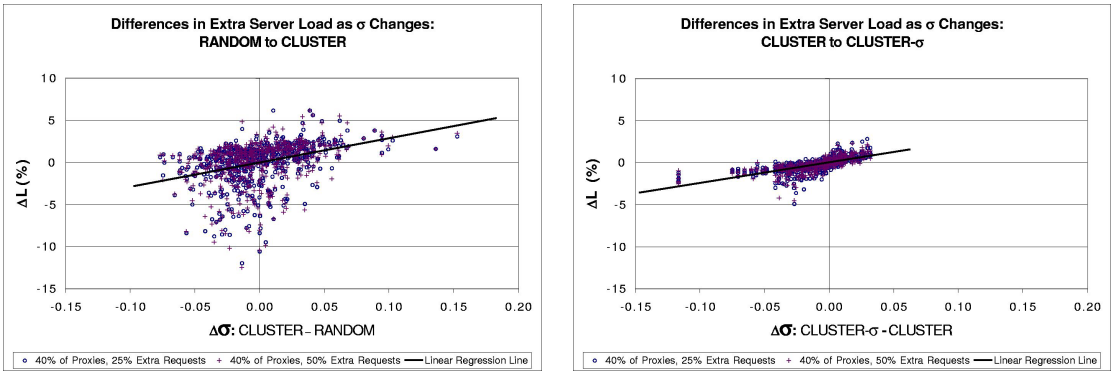


Figure 5: Differences in percentage of extra server load, ΔL, as N-values change between the CLUSTER and RANDOM strategies.



(a) (b)

Figure 6: Differences in the percentage of extra server load, ΔL, as average server σ changes between (a) the RANDOM and CLUSTER systems and (b) the CLUSTER and CLUSTER-σ systems.

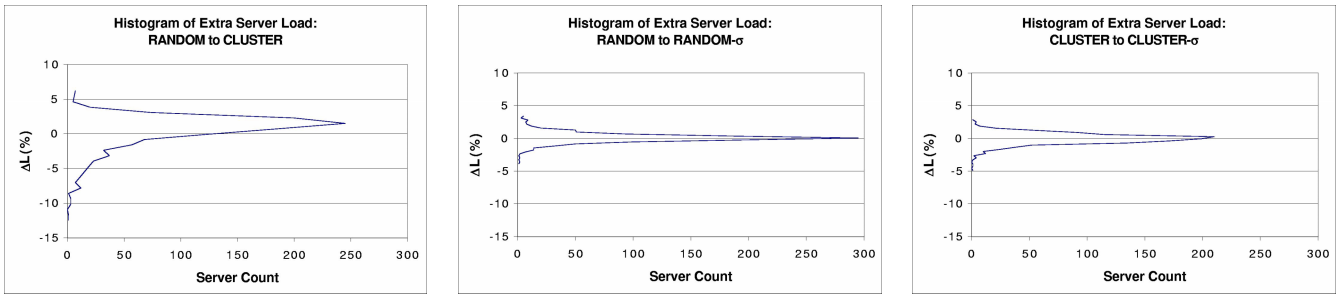
5.1 Questions and Answers

Question One: *If a server's N-value increases or decreases between different mapping strategies, does the extra read load experienced by the server also increase or decrease?*

Figure 5 displays the difference in the percentage of extra read load, ΔL, experienced by servers as their N-values improve under different mapping strategies. The x-axis shows the difference in a server's N-value, ΔN, when cluster-mapped (CLUSTER systems) versus randomly-mapped (RANDOM systems). Each data point represents a server. We see that a positive (negative) move in a server's N-value causes a likewise change in extra load at the server. In all experiments, the correlation in movement in N and extra load is between 83% and 86%. This behavior indicates that N, a function of the grid structure, has a strong influence on where extra load is distributed.

Question Two: *If the σs of a server's read sets increase or decrease under different proxy strategies, does the extra read load experienced by the server also increase or decrease?*

Yes, however, the trend is not as pronounced as it was for N. Figures 6a and 6b show ΔL, as a function of the difference, $\Delta\sigma$, in the average σ-values of a server's read-sets (average server σ), under differently formulated proxy strategies. Figure 6a compares RANDOM and CLUSTER systems whereas Fig. 6b compares CLUSTER and CLUSTER-σ systems. Note that the tilt of the regression trend-lines in both of these figures is less than the tilt caused by changes in N (Fig. 5), suggesting that the trend is not as strong as it was for N. Indeed, the correlation between changes in average server σ and extra load ranged from only 61% to 65% in all experiments. Furthermore, in Figs. 6a and 6b the data points are spread to all four quadrants of the graph, whereas in Fig. 5 data points are mostly in the all-positive or all-negative quadrants. This suggests that N has a more pronounced effect than σ. Note



(a) (b) (c)

Figure 7: Histograms showing differences in percentage of extra server load, ΔL, between (a) the CLUSTER and RANDOM systems (b) the RANDOM-σ and RANDOM systems, and (c) the CLUSTER-σ and CLUSTER systems.

that the data points are more widely spread in Fig. 6a than in 6b indicating that σ's effect on extra load distribution is more visible only after the grid is already optimized for N.

Question Three: *How is extra read load distributed in systems with good N-values versus systems with bad N-values?*

The answer to this question can already been seen in Figs. 5 and 6a which compare extra loads in randomly-mapped grids (large N errors) with those of clustered grids (small N errors). In Fig. 6a, it appears that the data points are mostly in the $\Delta L \geq 0$ region. A histogram showing the distribution of ΔL values confirms this observation (Fig. 7a). When cluster mapping is used, a large portion (an average of 63% across all experiments) of the servers get higher amounts of extra load than when randomly mapped. This indicates that systems with good N-values more evenly distribute extra load to servers.

Question Four: *How is extra read load distributed in systems that are σ-optimized versus non-σ-optimized?*

Load distribution is minimally affected by optimizing for σ. When comparing randomly-mapped grids before and after σ optimization, extra load changes for a server range from -4% to 3.5% (Fig. 7b). Similarly for clustered grids, the extra load changes range from -5% to 3% (Fig. 7c). Comparatively, load shifts in the range of -12.5% to 6% were observed for randomly mapped grids versus clustered grids without optimizing for σ (Fig. 7a). Thus, the shift in load is due mainly to changes in server N-values. Combined with the behavior seen in the answers to Questions One through Three, we conclude that grid structure, as captured by N-values, and not proxy strategies, which determine σs, is the main factor in where extra read load is distributed.

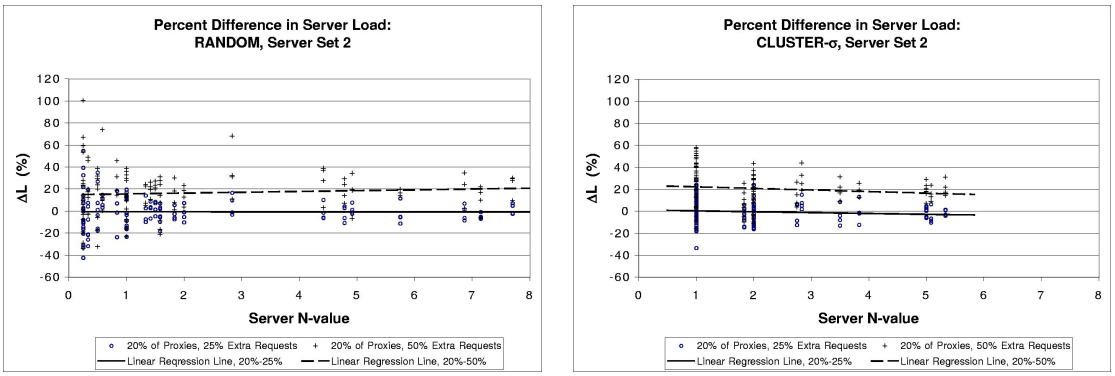
Also, note that unlike in Fig. 7a, the histograms in Figs. 7b and 7c are centered on zero load difference. This means that the overall effect of σ optimization on extra load distribution is minimal.

Question Five: *What does a server's N-value indicate about its extra read load?*

We have already partially answered this in Questions One and Three, but provide additional analysis here. Figure 8 shows the percentage of extra read load, ΔL, experienced by servers for the 20%-25% and 20%-50% initial-request-combinations (IR-combos). Figure 8a shows results for the RANDOM strategy and Fig. 8b shows the CLUSTER-σ strategy. Trend lines help distinguish between the two IR-combos. In both graphs we see that smaller N-values have highly variable extra read loads. This implies that systems could be made robust to variances in initial-request loads by making N-values large. However, increasing N-values would negatively affect load fairness.

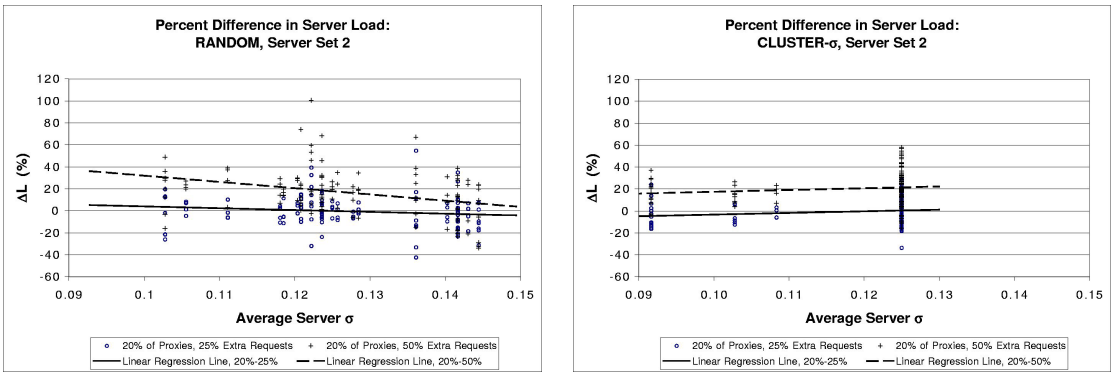
Note also that extra read load hardly changes from that of a baseline system for the 20%-25% proxy/load combination in Fig. 8 (i.e., the linear regression line lies right on the zero ΔL line). However, read loads jump noticeably for the 20%-50% combination. Since this behavior appears regardless of mapping strategy, this suggests that the grid structure and read protocol has a good deal of resilience to variations in initial-request loads and that beyond a certain threshold performance significantly degrades.

Question Six: *What do the σs of a server's read-sets indicate about the server's extra read load?*



(a) (b)

Figure 8: Percent differences in server loads plotted by server N-values for the second server set's (a) RANDOM system and (b) CLUSTER-σ system.



(a) (b)

Figure 9: Percent differences in server loads plotted by average server σ for the second server set's (a) RANDOM system and (b) the CLUSTER-σ system.

Results, arranged by the average σ-values of a server's read-sets (average server σ) for the second server set's RANDOM system are in Fig. 9a and results for the second server set's CLUSTER-σ system are in Fig. 9b. In both figures, we see that the largest swings in extra read load, ΔL, occur when average server σ equals the ideal σ for the grid (which is 0.125) and N is the sole factor in server loading. The RANDOM system also has large differences in extra load for other average σs (most notably at 0.136). This is due to the fact that N errors are large in the randomly-mapped systems, making σ nearly irrelevant to server loading.

Notice that in the RANDOM system extra load decreases as average server σ increases, which is counter to our expectations. The explanation for this behavior is that, for this particular server set. the proxies receiving extra initial-requests all have average server σs that are less than the ideal of 0.125. Therefore, the extra read requests are directed mostly to read-sets with less than ideal σ causing the servers in those read-sets to receive the bulk of the extra load.

5.2 Summary of Results

Overall, the experiments reveal the following notable observations about the proposed replication system when user request patterns deviate from that for which a system was originally designed.

- N, which is a factor of the grid structure, has the greatest influence on where extra server load is distributed. σ can also have an effect. However, σ's impact is often masked by N.
- Systems with good N-values more evenly distribute extra load to servers than do systems with bad N-values.

- Servers in systems built according to our construction method and operating protocols can be insulated from read load increases when initial-request loads increase at a subset of the proxies.

6 Conclusion

In [10] we proposed and validated an approach for developing replication systems which specialize in hosting large objects, or where user requests have long service times, or both. The replication strategy attempts to prevent servers from overloading by fairly distributing read loads to servers based on their relative capacities. Our approach to developing a fair replication strategy depends on optimizing two key parameters, σ and N. In [10] we showed that systems designed according to our approach are highly load-fair when user request patterns (the arrival rate of users' initial-requests) to servers in the system match that for which the system was designed.

In this paper we further studied the performance of systems constructed with our approach. We showed the influence σ and N have on server load when user request patterns vary from that for which a system was designed. Experimental results indicate that system structure, captured by N, has the most influence on load distribution. σ also affects load distribution, but not as much as N. We also saw that server loading is relatively unaffected by mild to moderate shifts in user request patterns.

References

[1] J. M. Almeida, D. L. Eager, and M. K. Vernon. A hybrid caching strategy for streaming media files. In *MMCN*, pages 200–212, 2001.

[2] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *IEEE Trans on Knowledge and Data Eng*, 4(6):582–592, 1992.

[3] I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. In *INFOCOM*, pages 1773–1780, 2001.

[4] J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, 1995.

[5] S. Ghandeharizadeh and R. R. Muntz. Design and implementation of scalable continuous media servers. *Parallel Computing*, 24(1):91–122, 1998.

[6] K. Guo, M. M. Buddhikot, Y. Chae, and S. Suri. Rcache: Design and analysis of scalable, fault tolerant multimedia stream caching schemes. In *Scalability and Traffic Control in IP Networks*, 2001.

[7] S. Harizopoulos, C. Harizakis, and P. Triantafillou. Hierarchical caching and prefetching for high performance continuous media servers with smart disks. *IEEE Concurrency*, 8(3):16–22, 2000.

[8] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *SIGCOMM*, pages 89–100, 1997.

[9] A. Kumar, M. Rabinovich, and R. K. Sinha. A performance study of general grid structures for replicated data. In *ICDCS*, pages 178–185, 1993.

[10] C. Mayer, K. S. Candan, and V. Sangam. Constraints, parameters, and strategies for replicating large content for web delivery. Technical Report TR-02-007, Computer Science and Engineering Department, Arizona State University, 2002.

[11] S. Sen, J. Rexford, and D. F. Towsley. Proxy prefix caching for multimedia streams. In *INFOCOM*, pages 1310–1319, 1999.

[12] C. Shahabi, M. Alshayeji, and S. Wang. A redundant hierarchical structure for a distributed continuous media server. In *IDMS*, pages 51–664, 1997.

[13] C. Vassilakis, M. Paterakis, and P. Triantafillou. Video placement and configuration of distributed video systems based on cable TV networks. *Multimedia Systems*, 8(2):92–104, 2000.

[14] B. Wang, S. Sen, M. Adler, and D. Towsley. Optimal proxy cache allocation for efficient streaming media distribution. In *INFOCOM*, 2002.

[15] K.-L. Wu, P. S. Yu, and J. L. Wolf. Segment-based proxy caching of multimedia streams. In *WWW*, pages 36–44, 2001.