Constraints, Parameters, and Replication Strategies for Supporting Web Content Delivery

Chris Mayer, K. Selçuk Candan*, and Venkatesh Sangam
Computer Science and Engineering Department,
Arizona State University
Email: {chris.mayer,candan,venkatesh.sangam}@asu.edu

Abstract

A new class of network related services has become widely used in improving performance, response-time, and scalability of Web-based applications. The basic premise of these services is that by replicating the content, user requests may be served from a server that is in the network proximity of the user instead of routing it all the way to the origin server. Replication, however, comes with update overhead. Although this overhead may be acceptable if the replicated objects are small, when they are large, as in multimedia objects, the cost of the network resources required for reflecting the update onto all replica servers may be prohibitively large. Secondly, in such systems server failure is a function of the read load and the traditional assumption of failure independence does not hold. Furthermore, there usually is a difference between read and write failures. In this paper, we propose a novel quorum-based method to deal with these issues. We identify system constraints and parameters that govern the design of a quorum structure with good load balancing behaviors and analyze them empirically.

Keywords: Content delivery networks, media replication, load balancing, quorums

1 Introduction

Web site performance is a key differentiation point among companies eager to reach, attract, and keep customers. This performance is measured using various metrics, including system up-time, average response time, and the maximum number of simultaneous users. Low performance, such as slowdowns, can be devastating for content providers, as shown by recent studies [32], which indicate that even with response times of 12 seconds, web sites find 70% abandonment rates (Table 1). Slowdowns observed by major web sites, especially during their peak access times, demonstrate the difficulty companies face trying to handle large demand volumes. For e-commerce sites, such slowdowns mean that potential customers are turned away from the electronic stores even before they have a chance to enter and see the merchandise. Therefore, improving the scalability of web sites is essential to companies and e-commerce sites eager to reach, attract, and keep customers.

*Contact author

Download time	Abandonment rate
≤ 7 seconds	7%
8 seconds	30%
12 seconds	70%

Table 1: Relationship between the time required to download a page and the user abandonment rate

1.1 Content Distribution

Many e-commerce sites observe non-uniform request distributions; i.e., although most of the time the request load they have is manageable, on certain occasions (for example during Christmas for e-shops and during breaking news for news services) the load they receive surges to very high volumes. Consequently, for most companies, investing in local infrastructure that can handle peak demand volumes, while sitting idle most other times, is not economically meaningful. These companies usually opt for *server farm-* or *edge-based* commercial services to improve scalability. Consequently, a new class of network related services has become a main ingredient to improve performance, response-time, and scalability of Web-based applications. Several high-technology companies are competing feverishly with each other to establish network infrastructures refered to as a *content delivery networks (CDNs)* [11, 4, 16, 27, 12]. The key technology underlying all CDNs is the deployment of network-wide caches which replicate the content held by the origin server in different parts of the network: front-end caches, proxy caches, edge caches, and so on. The basic premise of this architecture is that by replicating the content, user requests for a specific content may be served from a cache that is in the network proximity of the user instead of routing it all the way to the origin server. In CDNs, since the traffic is redirected intelligently to an appropriate replica, the system can be protected from traffic surges and the users can observe fast response times. Furthermore, this approach can not only eliminate network delays, but it can also be used to distribute the load on the servers more effectively. The CDN technology introduced new challenges in object replication and replica placement [28, 5, 8, 22, 7].

CDNs deploy multitudes of servers. The content provided by the customers, i.e., by the companies that pay CDNs for quality of service, are published into these servers. The requests of the end users, then, are routed to the most appropriate servers depending on the current network and server conditions. In most current CDN deployments, transactional semantics of the applications is not captured; all updates (data publishing) are performed by the CDN. Furthermore, in most applications, it is not necessary to ensure that the mutual exclusion property holds.

Replication, i.e. publishing the content onto multiple sites, has two advantages. First of all, it prevents *single points of failures*. Secondly, it enables *distribution of the total system load* across a set of servers. As long as the distribution is performed in a balanced manner and there are enough servers to accommodate the total load in the system, replication can ensure that every request is served in a timely manner.

1.2 Replication

Replication, however, comes with an update overhead. In the simplest case, if there are n replicas of an object o, any update to this object must also be replicated on each one of the replicas. Although this may be acceptable if the replicated objects are small, when these objects are large, as in video objects, the cost of the network resources required for reflecting the update onto all n replica servers may be prohibitively large, unless appropriate multicasting models

are utilized. On the Internet, today, there are two approaches to multicasting: IP-multicasting and application level multicasting. Currently, IP-multicasting is not a commonly available service. Application level multicasting, however, is an expensive service provided by companies, such as FastForward Networks, and is not economically feasible for most companies.

Quorum systems are one way to replicate data. Traditionally, a quorum system is a set, $S = \{s_1, s_2, \ldots, s_n\}$, of servers where the following conditions hold:

- **(Read Quorums)** $\mathcal{S}_\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$, where $\forall_{i \leq k} \;\; r_i \in 2^S$;
- **(Write Quorums)** $\mathcal{S}_\mathcal{W} = \{w_1, w_2, \ldots, w_l\}$, where $\forall_{i \leq l} \;\; w_i \in 2^S$;
- **(Read/Write Quorum Intersection Property)** $\forall_{(i \leq k) \wedge (j \leq l)} \;\; r_i \cap w_j \neq \emptyset$; and
- **(Write/Write Intersection Property)** $\forall_{(i \leq l) \wedge (j \leq l)} \;\; w_i \cap w_j \neq \emptyset$.

When an object is to be updated, a write quorum, $w_j \in \mathcal{S}_\mathcal{W}$, is selected and the update is performed in *all* servers in w_j. When an object is to be read, a read quorum, $r_i \in \mathcal{S}_\mathcal{R}$, is selected and the object is read from *one of the* servers in r_i. Since every write quorum has a non-empty intersection with every read quorum, the read operation is guaranteed to find at least one server within the read quorum that contains an up-to-date copy of the object. The cost of a write operation is limited by the size of the write quorum, w_j, selected and the read load of the system is split between the read quorums. Furthermore, as long as the read and write quorums can be selected without communicating with a central authority, it is possible to have a completely non-central, distributed, management of the replicas.

As will be made clear later, we employ a quorum-based replication strategy as the basis for a CDN. In Section 10 we review related work in quorum systems and draw distinctions between the past work and quorum systems as employed in this paper.

1.3 Challenges

Server failures can cause the failure of some of the read requests in the system. It is possible that when a read quorum is chosen for the read operation, the only server which contains an up-to-date version of the object in that read quorum may be a failed server. Traditional quorum systems assume that the probability of server failure (representing a system failure, such as disk or CPU crash) is independent of other system parameters and the failure probabilities of other servers. For example, in a related work [9], Cheung *et al.* studied the load sharing behavior of the grid protocol in environments with high performance requirements and proposed a grid protocol with desirable properties. However, they assumed independent server failures and they fine tuned their protocol to have good transactional (such as mutual exclusion) behavior. As shown in Figure 1, however, in web servers, as well as in other server systems, failure is actually a function of the request load; i.e., a system can fail to satisfy the requests even though it is physically up and running.

The performance of a server, s_i, can usually be represented by its average response time and the maximum allowable request load, as shown in Figure 1. Beyond its capacity, the performance of the server deteriorates very quickly, essentially failing to serve the requests it receives. This performance characteristic is very common in disks and, especially, web servers, which can support only a limited number of simultaneous connections. Once this limit is exceeded, the response time of the server increases very quickly, rendering the server ineffective. Therefore, any web server whose load exceeds its maximum capacity can be considered, for all practical purposes, a failed server. Note also that,

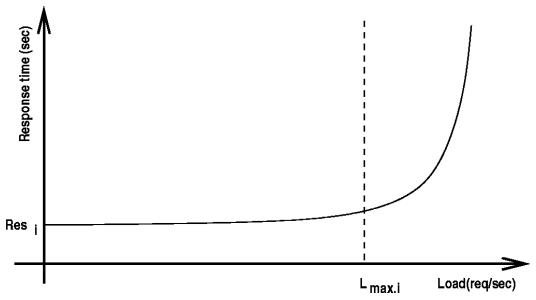


Figure 1: Performance of a server s_i. $L_{max,i}$ represents the capacity, and Res_i represents average response time

assuming that the number of requests in the system is pretty much constant over time, a failure in one of the servers in the system may cause an increase in the request load of the other servers in the system, thereby increasing their failure probability as well. Consequently, we can see that in web content distribution systems,

- *the traditional assumption of failure independence does not hold.* First of all, as the load in the system increases, the probability failure of all servers increase together; therefore the failure rate is not independent. Secondly, once a server is failed, the requests are likely to be redirected to the servers that are alive, increasing their loads and hence their failure probabilities.
- *there is a difference between read and write processes.* Since, most of the time, read (by the users) and write (by the CDN) processes use different mechanisms, an increased read load may or may not affect the write process. Therefore, it is possible that a server is read-failed while it can serve writes and vice versa. Also, write policies can be fine-tuned, whereas reads are randomly directed to servers by the users. Furthermore, since all writes are performed by the CDN, the Write/Write intersection property does not need to be enforced.

Since our aim is to minimize the load on the servers (to prevent them from failing), it is desirable to have large write quorums. However, since we also want to minimize the cost of writes, which may be prohibitive in the case of large media objects, it is desirable to have small write quorums. In this paper, we will assume that there is an upperbound on the number of servers that can be in a write quqorum. Given a set of servers, S, choosing the right set of write and read quorums is essential for preventing failures. In this paper, we identify system constraints and parameters that govern the design of appropriate quorum structures. We develop novel replication strategies, based on quorum systems, to address challenges introduced by the nature of the web content distribution problem. In next section, we start by defining the problem in more formal terms.

2 Problem Formulation

A major challenge in object replication using quorums is to identify the best way to select the read and write quorums to be used. As discussed above, the way these quorums are selected has a big impact on the performance (in terms of cost, response time, failure probability etc.) of the overall system. The assumptions and constraints that apply to the problem of media distribution over the web by CDNs are different from the ones that are used in traditional quorum work, necessitating the development of novel replication techniques.

2.1 Assumptions and Constraints

We can list the assumptions and constraints that govern the problem as follows:

Constraint I. The CDN employs a set of servers, $S = \{s_1, s_2, \ldots, s_n\}$, where the performance of each server, $s_i \in S$, is represented by a 2-tuple $p(s_i) = \langle L_{max,i}, Res_i \rangle$ (load capacity and average response time, respectively). In this paper, we will assume that, as long as the load on the servers are below their maximum load capacity, their response times are virtually identical; i.e., $\forall_{i,j \leq n}(Res_i = Res_j)$. Therefore, the real factor that differentiates servers from each other is the the load that they can accommodate. Note that, in a truly homogeneous setting, it is possible that all servers will also have the same capacity (i.e., $\forall_{i,j \leq n}(L_{max,i} = L_{max,j})$), however, we refrain ourselves from making such an assumption.

Constraint II. Write operations are performed by the CDNs; i.e., the CDN has a mechanism that chooses to which quorum a given object will be published. In this paper, we assume that the criteria for the selection of the quorums for the write operations will be to balance the resulting read load on the servers. Note that the average write time of the system is defined as

$$t_{write} = t_{write\ quorum\ selection} + t_{write\ all\ servers\ in\ quorum}.$$

For large media objects, $t_{write} \simeq t_{write\ all\ servers\ in\ quorum}$. Also, there is an upper bound, max_write, on the cost of write operations; i.e., the size of the largest write quorum that CDN employs. Finally, since all writes are performed by the CDN, the Write/Write intersection property does not need to be enforced.

Constraint III. Read operations are performed by the end-users through their browsers or their media players. It is the responsibility of the CDN to redirect these requests to most suitable server. Therefore, the read protocol is as follows:

1. Client chooses a proxy server and delegates the task of identifying an appropriate data server to the proxy
2. The proxy server selects a read quorum among the ones to which it belongs
3. The proxy server identifies the server with the most up-to-date copy of the object, either using a directory server or by communicating with the servers in the read quorum
4. The proxy server redirects the client to the appropriate server.

In this paper, we will assume that the probability with which initial, proxy servers are contacted by clients is uniformly distributed. This makes sense, as the clients are unaware of the quorum structures used by the CDN. As such, the average read time of the system can be defined as

$$t_{read} = t_{read\ quorum\ selection} + t_{read\ quorum\ access} + t_{read\ server\ selection} + t_{read\ server\ access},$$

where $t_{read\ quorum\ selection}$ is the average time to select a read quorum in $\mathcal{S}_{\mathcal{R}}$, $t_{read\ quorum\ access}$ is the average time to access the selected read quorum and identify which of those have the most recent replica, $t_{read\ server\ selection}$ is the average time to select the appropriate server among the candidate servers, and $t_{read\ server}$ is the average time required for accessing and retrieving the requested object from the selected server. When the retrieved objects are large media objects, $t_{read} \simeq t_{read\ server\ access}$.

Constraint IV. As we mentioned earlier, the traditional assumption of failure independence does not generally hold. When the load on a server increases beyond a threshold, it fails; therefore the failure rate is a function of the load. Furthermore, as the load in the system increases, the probability failure of all servers increase together; therefore the failure rate of the servers are not independent. Note that independent failures due to disk or CPU crashes are still possible.

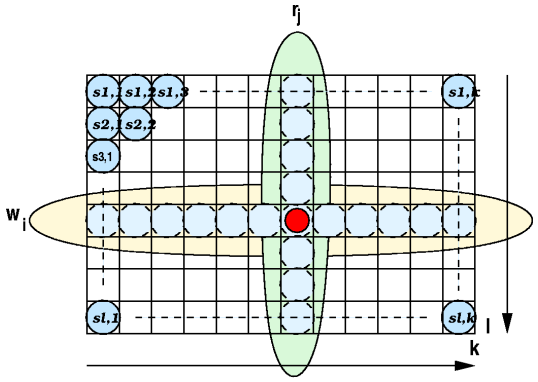


Figure 2: Grid structure guarantees that the intersection property holds

Constraint V. There is a difference between read and write failures. Since, most of the time, read and write processes use different mechanisms, an increased read load may or may not affect the write process. Therefore, it is possible that, a server is read-failed while it can serve writes by the CDN and vice versa. In this paper, we will focus on preventing the read failures. A read failure occurs if (a) the read quorum selection process, (b) the read server selection process, or (c)the read server access process fails. In this paper, we assume that the read quorum selection process fails only if the initial server is physically inaccessible. The read server selection and access processes however may fail if some of the involved servers are overloaded and not accepting additional requests.

2.2 Task of the CDN

Given the above assumptions and constraints, the task of the CDN is to identify a quorum system $\mathcal{Q} = \langle \mathcal{S}_{\mathcal{R}}, \mathcal{S}_{\mathcal{W}} \rangle$, such that the average failure rate and read time of the entire system is minimized. This task has two parts:

- Distributing the read load according to the server capacities. This will ensure that none of the servers is pushed close/beyond their capacities.
- Ensuring that the system is capable of handling more than 1, say k, disk failures. This will ensure that the user requests will be satisfied even if some of the servers in the system fail due to capacity overload or system failures.

Note that these tasks may conflict with each other; i.e., it may not be possible to achieve both tasks at the same time. In this paper, we will focus on the first task: (1) we formally describe the task as an optimization problem, (2) we identify parameters that capture the optimality of a solution, and (3) we report on experimental results that show the effects of these parameters. We are currently working on the second task.

2.3 Quorum System Selection

Many quorum based object replication algorithms use a grid structure to select read and write quorums [20, 9, 21, 29, 19, 24]. In the basic grid approach, the given set of servers, $S = \{s_1, s_2, \ldots, s_n\}$, are placed into the nodes of a grid of size $l \times k (= n)$, where

- each of the l rows correspond to a write quorum: $\mathcal{S}_{\mathcal{W}} = \{w_1, w_2, \ldots, w_l\}$, where $\forall_{i \leq l} \;\; w_i \in 2^S$ and
- each of the k columns correspond to a read quorum: $\mathcal{S}_{\mathcal{R}} = \{r_1, r_2, \ldots, r_k\}$, where $\forall_{i \leq k} \;\; r_i \in 2^S$.

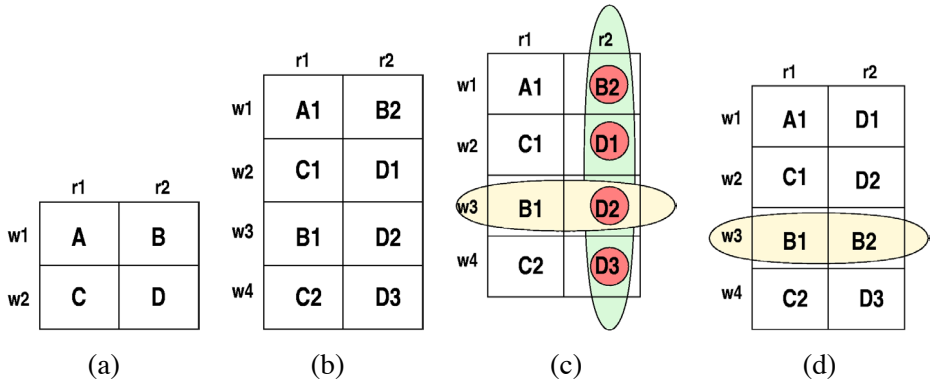


Figure 3: Various ways to mapping four servers onto grid structures

As shown in Figure 2, this structure guarantees that the intersection property holds; i.e. $\forall_{(i \leq k) \wedge (j \leq l)} \quad r_i \cap w_i \neq \emptyset$. Note that the grid-based quorum systems proposed elsewhere are variants of the approach just presented and are aimed to increasing system availability, assuming independent server failure probabilities. In this paper, we formulate our solution using this basic definition of a grid quorum structure. However, in order to capture the specific requirements of the problem we are trying to solve, we augment the basic algorithm and protocol. Given this protocol, we would like to distribute the overall *read* load on the servers in a way that reflects the maximum number of simultaneous requests they can handle. In the next section, we will discuss the factors affecting server load distribution.

3 Server Read Load Distribution

It is not hard to see that an arbitrary mapping of servers onto a grid will not distribute the load relative to server capacities. Let us consider the following example:

Example 3.1 Let us be given four servers, A, B, C, and D. Let us assume that $L_{max,A} = 100$, $L_{max,B} = 200$, $L_{max,C} = 200$, and $L_{max,D} = 300$. For the sake of this example, let us assume that these servers are placed onto a grid as shown in Figure 3(a). Assuming that writes are distributed uniformly across the two write quorums w_1 and w_2 and that initial contacts are uniformly distributed across all four servers, we can find the total load on these servers as follows:

- The read load on server A can be calculated as follows: Let us assume that the total read load in the system is L. Then, on the average, $I_A = \frac{L}{4}$ of these requests will initially access server A and $I_C = \frac{L}{4}$ of these requests will initially access server C. Again, on the average, half of the requests that initially arrive at server A will need to be served by A and the other half will need to be redirected to C. Similarly, half of the requests that initially arrive at server C will need to be served by C and the other half will need to be redirected to A. Consequently, the total load on server A is

$$I_A * 0.5 + I_C * 0.5 = \frac{L}{4} * 0.5 + \frac{L}{4} * 0.5 = \frac{L}{4}.$$

- The read loads on the other servers, B, C, and D, can also be calculated in the same way. In this example, they are each equal to $\frac{L}{4}$.

This means that, if this particular server to grid mapping is used, the overall read load will be uniformly distributed among the available servers. However, this means that as the overall load increases, server A which has a lower capacity compared to the other servers, will become overloaded while other servers still have enough resources to accommodate more requests. Consequently, the system is limited by the capacity of the weakest server in the system. Clearly, in this example, a more desirable load distribution would be proportional the capacities of the servers:

$$L_A = \frac{L}{8}, L_B = \frac{2L}{8}, L_C = \frac{2L}{8}, \text{ and } L_B = \frac{3L}{8}.$$

This distribution would ensure that the load is distributed according to the capacities of the servers, thereby preventing the weakest server from becoming a bottleneck. ◇

4 Partitioning and Virtual Servers

One intuitive way to achieve a more fine-grained capacity distribution is to split/partition each server into multiple servers. Note that fragmentation of data has been proposed for reducing storage requirements in grid structures [1]. This is not the same thing we are proposing. Our focus in this paper is to use partitioning to achieve a desirable load balancing behavior. That is, given a set of servers $S = \{s_1, s_2, \ldots, s_n\}$, where (a) each server s_i has a maximum capacity of $L_{max,i}$, and (b) the weakest server in the set has a capacity of L_{weak}, we can split each server s_i into $n_i = \lfloor \frac{L_{max,i}}{L_{weak}} \rfloor$ *virtual* servers, $V(s_i) = \{s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}\}$, and place them onto a larger grid.

Example 4.1 Let us reconsider the servers, A, B, C, and D, in the previous example. Since, we have $L_{max,A} = 100$, $L_{max,B} = 200$, $L_{max,C} = 200$, and $L_{max,D} = 300$, the weakest server, A, has a capacity of, $L_{weak} = 100$. Consequently, we can split A into 1, B and C each into 2, and D into 3 virtual servers. Figure 3(b) shows one possible mapping of these virtual servers onto a grid with 2 columns; i.e. with write quorums of size two. ◇

In the context of grid structures, partitioning of servers can have various effects on the performance. These effects have to be studied carefully to prevent unexpected consequences.

Multiple quorum intersections. One major consequence of this new mapping strategy is that a write quorum and a read quorum can intersect at multiple cells of the grid. For example, as shown in Figure 3(c), the write quorum w_3 which involves servers B and D intersects the read quorum r_2 at all four cells of the corresponding column. Consequently, if an object is written to write quorum w_2, and read quorum r_2 is chosen for the read operation, server D is 3 times as likely as server B to serve the object.

Write load as a function of the read capacity. Another consequence of this strategy is that, assuming that the write quorums are selected in an evenly distributed manner, each server will have an object write load that is a function of its read capacity. In this example, server A is in 1 write quorum, server B is in 2 write quorums, server C is in 2 write quorums, and server D is in 3 write quorums. Note, however, that it is also possible to have more than one virtual server of the same physical server be placed on the same write quorum. In such a case, as in Figure 3(d), a server may have a lower object write load relative to its read capacity: server A is in 1 write quorum, **server B is in 1 write quorum**, server C is in 2 write quorums, and server D is in 3 write quorums.

Uneven read quorum selection. Since a server may have two or more virtual servers each in a different read quorum, when a server receives an initial request for an object, it has to choose which of the read quorums it is involved in

will be used for delivering the object. For instance, in both Figures 3(c) and (d), server B has virtual servers in read quorums r_1 and r_2. Therefore, server B can choose either one of the read quorums to serve a given request (we will study the effects of this selection in Section 6). If on the other hand, server A receives a request, then the only read quorum it can use for serving the object is r_1.

Multiple quorum intersections and uneven read quorum selection properties complicate the load balancing task. In the next section, we formally study the effects of these properties on the load distribution.

5 Constraints Governing the Server-to-Grid Mapping

We can intuitively see that the read load of a server can be controlled by selecting an appropriate server-to-grid mapping. Before discussing how to select mappings, in this section, we discuss the constraints that govern the loads on the servers. This section introducea relevant terminology, provides appropriate tools and parameters to define mappings, and sheds light on the solutions we develop in the following sections. Let us have

- a set of servers $S = \{s_1, s_2, \ldots, s_n\}$, where each server s_i is split into a set of virtual servers, $V(s_i) = \{s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}\}$. Let V be the set of all virtual servers;
- a grid $G_{l \times k} = \{g_{1,1}, \ldots, g_{l,1}, g_{1,2}, \ldots, g_{l,k}\}$ which denotes the set of cells of a grid with l rows and k columns; and
- a mapping $\mu : V \rightarrow G_{l \times k}$, that maps each one of the virtual servers into a cell of a grid, such that no two virtual servers occupy the same cell and all cells are occupied by a virtual server.

Given a total system read load, L, the read load of server s_i can be found as follows:

$$\begin{aligned} L(s_i) \;=\; & L \times \sum_{1 \le t \le k} \sum_{1 \le p \le l} prob(w_p \text{ contains the requested object}) \times \\ & prob(r_t \text{ is chosen for read} \mid w_p \text{ contains the requested object}) \times \\ & prob(s_i \text{ is selected to serve the object} \mid r_t \text{ is chosen for read} \wedge \\ & \qquad w_p \text{ contains object}) \end{aligned} \tag{1}$$

In Equation 1, w_p denotes a write quorum and r_t denotes a read quorum. Since we assume that the read quorum selection is performed in a way that is unaware of which write quorum has an updated copy of the object, the read quorum selection is independent of the write quorum that contains the object. We can rewrite Equation 1 as

$$\begin{aligned} L(s_i) \;=\; & L \times \sum_{1 \le t \le k} prob(r_t \text{ is chosen for read}) \times \\ & \sum_{1 \le p \le l} prob(w_p \text{ contains the requested object}) \times \\ & prob(s_i \text{ is selected to serve the object} \mid r_t \text{ is chosen for read} \wedge \\ & \qquad w_p \text{ contains object}) \end{aligned} \tag{2}$$

Furthermore, since we assume that updates are distributed uniformly across all write quorums, we can further refine the equation as

$$\begin{aligned} L(s_i) \;=\; & \frac{L}{l} \times \sum_{1 \le t \le k} prob(r_t \text{ is chosen for read}) \times \end{aligned}$$

$$\sum_{1 \leq p \leq l} prob(s_i \text{ is selected to serve the object} \mid r_t \text{ is chosen for read} \wedge w_p \text{ contains object}) \quad (3)$$

Note that, in general, we would like to ensure that the mapping, μ, chosen by the CDN, distributes the total read load onto individual servers, relative to their read capacities:

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

Assuming that μ is chosen to ensure that this condition holds, we can rewrite Equation 3 as

$$\frac{n_i}{k} = \sum_{1 \leq t \leq k} prob(r_t \text{ is chosen for read}) \times \sum_{1 \leq p \leq l} prob(s_i \text{ is selected to serve the object} \mid r_t \text{ is chosen for read} \wedge w_p \text{ contains object}) \quad (4)$$

Equation 4 gives us the main condition that the CDN should observe while choosing a mapping, μ. Although it does not specify how an appropriate mapping can be constructed, we can benefit from this constraint to extract properties of good mappings. But, first, we need to introduce some shorthand to simplify the rest of the discussion. Let

- S_{w_p} be the set of servers that have virtual servers in write quorum, w_p; i.e., $S_{w_p} = \{s_i | V(s_i) = g_{p,*}\}$;
- S_{r_t} be the set of servers that have virtual servers in read quorum, r_t; i.e., $S_{r_t} = \{s_i | V(s_i) = g_{*,t}\}$;
- $V_{p,t}$ be the set of virtual servers in read quorum t that have a corresponding server in write quorum p; i.e., $V_{p,t} = \{v_j \mid v_j \in V(s_i) \wedge \mu(v_j) = g_{*,t} \wedge s_i \in S_{w_p}\}$.

Then, if an object is in write quorum, w_p, and a read quorum r_t is chosen for the read operation, assuming that servers are selected as a function of their presence in the read quorums, the probability of s_i being selected for the read operation will be equal to $\frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|}$. Hence, we have

$$\forall s_i \in S, \quad \frac{n_i}{k} = \sum_{1 \leq t \leq k} prob(r_t \text{ is chosen for read}) \times \sum_{1 \leq p \leq l} \frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|}, \quad (5)$$

Finding a mapping, μ, that satisfies the above constraint for all servers is not trivial. However, we can simplify this task by carefully reducing the size of the search space using appropriate assumptions. In Section 9.1, we will study the effects of these assumptions and show that they can be relaxed without significantly affecting the quality of the result.

Simplifying Assumption 1 Let us assume that, we can choose μ such that

$$\forall_{1 \leq t \leq k},\ prob(r_t \text{ is chosen for read}) = \frac{1}{k}$$

◇

Therefore, the first parameter we will consider is *the read quorum load distribution*. We will denote this parameter as $\sigma(r_t)$, where r_t is a read quorum, and we state that, for all read quorums, the ideal value for $\sigma(r_t)$ is $\frac{1}{k}$. Note that, as we informally discussed in Section 4, it is not trivial to ensure that this assumption holds. In Section 6, we will formally study the properties of parameter σ and in Section 9.1 we will experimentally evaluate its effect on the

optimality of grid mappings. For now, though, we will assume that this simplifying assumption holds. Then, we can see that Equation 5 reduces to,

$$\forall s_i \in S, \sum_{1 \leq t \leq k} \sum_{1 \leq p \leq l} \times \frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|} = n_i. \quad (6)$$

Although this assumption reduces the solution space significantly, finding a mapping, μ, that satisfies Equation 6 for all servers is not trivial either. However, if, in addition to the simplifying assumption 1, we assert the following assumption, then the CDN can guarantee a desirable server load distribution.

Simplifying Assumption 2 $\forall_{s_i \in S} \forall_{1 \leq t \leq k} \forall_{1 \leq p \leq l} \frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|} = \frac{n_i}{k \times l}$ holds. ◇

To see that the simplifying assumption 2 is a sufficient condition for Equation 6, consider the following:

$$\sum_{1 \leq t \leq k} \sum_{1 \leq p \leq l} \times \frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|} = \sum_{1 \leq t \leq k} \sum_{1 \leq p \leq l} \frac{n_i}{k \times l} = \frac{n_i}{k \times l} \times \sum_{1 \leq t \leq k} \sum_{1 \leq p \leq l} 1 = \frac{n_i}{k \times l} \times k \times l = n_i.$$

Therefore, the second parameter we will consider is *the read and write quorum intersection*. We will denote this parameter as $N(s_i) = \sum_{1 \leq t \leq k} \sum_{1 \leq p \leq l} \times \frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|}$, where s_i is a server. We state that, for all servers, the ideal value for $N(s_i)$ is n_i.

Note that, although the simplifying assumptions enable us to reduce the search space significantly by allowing us concentrate on two parameters, *read quorum load distribution* and *read and write quorum intersection*, we need to further study these parameters to understand their exact effects on the server-to-grid mapping and whether they can be used for developing efficient mapping strategies. In the next sections, we will study these assumptions and the conditions under which they are expected to hold in greater detail.

6 Properties of σ: Read Quorum Load

When a server, which has two or more virtual servers, each placed in a different read quorum, receives an initial request for an object, it has to choose one of the read quorums for delivering the object. Since at the time when it receives the request, the initial server may not know which write quorum or which servers have the most up-to-date copy of the object being requested, it can not use *a priori* knowledge to select the most suitable read quorum among the candidates. In this section, we will present two strategies that an initial server can use to choose a read quorum without any external knowledge, and we will calculate the resulting read quorum loads.

6.1 Calculating σ under Weighted Server Participation Strategy

The first strategy that initial server, s_i, can use is to assume that, *since the client initially accessed server, s_i, it would prefer to receive the object from server s_i if possible*. Although s_i does not know whether it has the latest version of the object or not, it can maximize its own chances of serving the object by giving preference to the read quorums that it has high participation (i.e., more virtual servers of s_i) over those read quorums that it has low participation. In order to implement this strategy, the only information s_i needs is the list of read quorums in which its virtual servers are located and, for each such read quorum, the number of its virtual servers located in it.

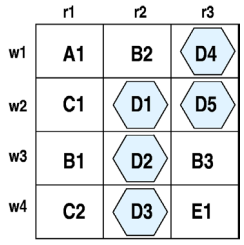


Figure 4: An example quorum structure with varying server participations on read quorums

Example 6.1 Let us consider the quorum structure given in Figure 4 and let us assume that a client initially accessed server D to request an object. Server D is present in two read quorums, r_2, and r_3: its participation in r_2 is with 3 virtual servers and its participation in r_3 is with 2 virtual servers. Consequently, under this strategy, D will select read quorum, r_2, with $\frac{3}{5}$ probability and read quorum, r_3, with $\frac{2}{5}$ probability. $\diamond$

We can generalize this example as follows: Given an initial server s_i which has a set of virtual servers, $V(s_i) = \{s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}\}$ and a mapping $\mu : V \rightarrow G_{l \times k}$, we can calculate the likelihood of a read quorum r_j to serve the request as follows:

$$prob(r_j \text{ is chosen for read} \mid \text{client initially accessed server } s_i) = \frac{|\{u|(v_p \in V(s_i) \wedge (\mu(v_p) = g_{u,j})\}|}{n_i}$$

Consequently, the total load on a given read quorum, r_j, due to the requests initially received by server, s_i, can be calculated as

$$\frac{L}{n} \times \frac{|\{u|(v_p \in V(s_i) \wedge (\mu(v_p) = g_{u,j})\}|}{n_i} \quad or \quad \frac{L}{n \times n_i} \times (\text{number of virtual servers of } s_i \text{ in read quorum } r_j).$$

Hence, we can calculate the total read load, $L_w(r_j)$, of a given read quorum, r_j, under weighted server participation and the parameter $\sigma_w(r_j)$ ($\frac{L_w(r_j)}{L}$) as

$$L_w(r_j) = \sum_{\{v_p|\mu(v_p)=g_{*,j}\}} [v_p \in V(s_i)] \times \frac{L}{n \times n_i} \quad \text{and} \quad \sigma_w(r_j) = \sum_{\{v_p|\mu(v_p)=g_{*,j}\}} \frac{[v_p \in V(s_i)]}{n \times n_i} \qquad (7)$$

respectively.

6.2 Calculating σ under Flat Server Participation Strategy

Note that the initial server, s_i, can further refrain itself from making any assumptions. In this case, it gives no preference to any of the quorums it is involved in; i.e., it randomly chooses among all the quorums that it belongs to. In order to implement this strategy, the only information s_i needs is the list of read quorums in which it has virtual servers.

Example 6.2 Let us reconsider the quorum structure given in Figure 4 and again assume that a client initially accessed server D to request an object. Server D is present in two read quorums, r_2, and r_3. Consequently, irrespective of its varying participation in these two read quorums, under this strategy, D will select read quorum, r_2, with $\frac{1}{2}$ probability and read quorum, r_3, with $\frac{1}{2}$ probability. $\diamond$

We can generalize this as follows: Given a server s_i which has a set of virtual servers, $V(s_i) = \{s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}\}$ and a mapping $\mu : V \rightarrow G_{l \times k}$, we can calculate the total load on a given read quorum, r_j, due to the requests initially received by server, s_i, as

$$prob(r_j \text{ is chosen for read} \mid \text{client initially accessed server } s_i) = \frac{[\exists_{v_p \in V(s_i)}(\mu(v_p) = g_{*,j})]}{|\{h | v_p \in V(s_i) \wedge (\mu(v_p) = g_{*,h})\}|}$$

Consequently, the total load on a given read quorum, r_j, due to the requests initially received by server, s_i, can be calculated as

$$\frac{L}{n} \times \frac{[\exists_{v_p \in V(s_i)}(\mu(v_p) = g_{*,j})]}{|\{h | v_p \in V(s_i) \wedge (\mu(v_p) = g_{*,h})\}|}$$

or, equivalently, as

$$\begin{cases} 0 & \text{if } r_j \text{ does not contain a virtual server of } s_i \\ \dfrac{L}{n \times |\{h | v_p \in V(s_i) \wedge (\mu(v_p) = g_{*,h})\}|} & \text{if } r_j \text{ contains a virtual server of } s_i \end{cases}$$

This means that we can calculate the total read load of a given, $L_f(r_j)$, read quorum, r_j, under flat server participation as

$$L_f(r_j) = \sum_{\{s_i | v_h \in V(s_i) \wedge \mu(v_h) = g_{*,j}\}} \frac{L}{n \times |\{u | v_p \in V(s_i) \wedge (\mu(v_p) = g_{*,u})\}|}.$$

Therefore, the value of the parameter $\sigma_f(r_j) = \frac{L_f(r_j)}{L}$, under flat server participation strategy can be calculated as

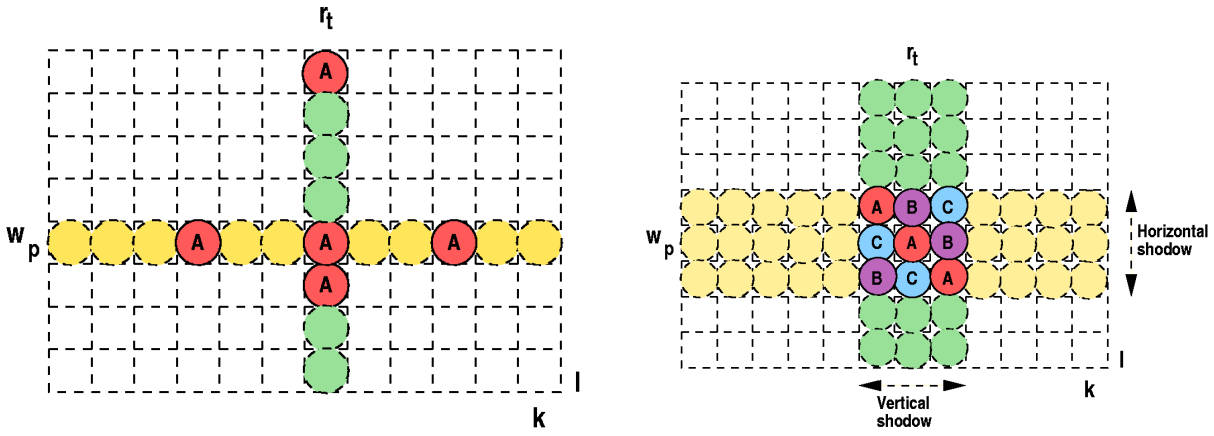
$$\sigma_f(r_j) = \sum_{\{s_i | v_h \in V(s_i) \wedge \mu(v_h) = g_{*,j}\}} \frac{1}{n \times |\{u | v_p \in V(s_i) \wedge (\mu(v_p) = g_{*,u})\}|}. \quad (8)$$

6.3 Summary

Both strategies are very simple to implement in a distributed environment. However, finding a mapping that will provide evenly distributed access rates for the read quorums using these strategies is not trivial. Fortunately, as Section 9.5 explains, having evenly distributed read quorum load is not totally essential to low server load.

7 Properties of N: Read and Write Quorum Intersection

In Section 5, we stated that, if an object is placed in write quorum, w_p, and a read quorum r_t is chosen for the read operation, then *assuming that servers are selected as a function of their presence in the read quorums,* the probability of s_i being selected for the read operation will be equal to $\frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|}$. Note that this assumes that it is possible for the proxy server to know (1) which write quorum was the object placed in and (2) what is the virtual server intersection between the corresponding read and write quorums. This can be achieved as follows: for every cell location, the intersection, $V_{p,t}$, between the corresponding read and write quorums are computed at the mapping time and this information is written into the servers. Each time an object is published into a write quorum, the ID (row number)



(a) (b)

Figure 5: (a) Each write quorum and read quorum intersects at a single server, and (b) we can extend the result from one single server to a set of servers (the diagonal pattern is not necessary). The shaded areas in the figure are the *horizontal* and *vertical shadows* of the cluster.

of the quorum is also written along with the timestamp. Then, when a read quorum is chosen to serve the content, the virtual server located at the intersection cell sends the $V_{p,t}$ information to the proxy. This way, the entire mapping information does not need to be known globally by all servers, and the $V_{p,t}$ information can be recovered in a distributed manner. With this understanding, there are various ways to ensure that simplifying assumption 2,

$$\forall_{s_i \in S} \forall_{1 \leq t \leq k} \forall_{1 \leq p \leq l} \frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|} = \frac{n_i}{k \times l},$$

holds. In this section, we try to identify these ways using constraints imposed on the intersections of quorums.

7.1 Single Server Intersection Constraint

One way to ensure that the second simlifying assumption holds is to require that each write quorum and read quorum intersects at a single server $s_i \in S$ (Figure 5(a)):

$$\forall_{s_i \in S} \forall_{1 \leq t \leq k} \forall_{1 \leq p \leq l} (V(s_i) \cap V_{p,t} = V_{p,t}) \vee (V(s_i) \cap V_{p,t} = \emptyset);$$

or, equivalently,

$$\forall_{1 \leq t \leq k} \forall_{1 \leq p \leq l} |S_{w_p} \cap S_{r_t}| = 1.$$

7.2 Set Intersection Constraint

Single server intersection constraint, however, can be very over restrictive in the design of quorum systems. We can generalize it to sets of servers. Let us assume that we are given a set of servers S_0, such that each server in S_0 has the same number of virtual servers as the others in S_0. Let also $V(S_0)$ denote $\cup_{s_j \in S_0} V(s_j)$. Then, if

$$\forall_{1 \leq t \leq k} \forall_{1 \leq p \leq l} (V(S_0) \cap V_{p,t} = V_{p,t}) \vee (V(S_0) \cap V_{p,t} = \emptyset)$$

or, equivalently,

$$\forall_{1\leq t\leq k}\forall_{1\leq p\leq l}(S_{w_p}\cap S_{r_t}=S_0)\vee(S_{w_p}\cap S_{r_t}\cap S_0=\emptyset),$$

then the constraint holds for all servers in S_0. This means that, we can extend the constraint from one single server to a set of servers. The consequence of this constraint is that, as shown in Figure 5(b), given a cluster of virtual servers, $V(S_0)$, that are placed onto the grid, the *horizontal and vertical shadows* of the cluster (as shown in Figure 5(b)) are constrained in terms of the servers that can be placed without violating the constraint.

7.3 Optimal Cluster Dimensions based on the Set Intersection Constraint

This constraint can be used for reducing the search space. The optimal dimensions of a server cluster of dimensions $x\times y$ can be found by minimizing the product of its horizontal and vertical shodows. Intuitively, for every grid position in the horizontal shadow, all grid positions in the vertical shadow are constrained. Therefore, we can find optimal dimensions, x and y, by solving

$$minimize\{((k-x)\times y)\times((l-y)\times x))\},$$

where, x is the number of read quorums covered by the cluster, y is the number of write quorums covered by it, and $x\times y=|V(S_0)|$. It can be shown that the optimal cluster dimensions are

$$x\simeq\sqrt{\frac{k}{l}\times|V(S_0)|}\qquad and\qquad y\simeq\sqrt{\frac{l}{k}\times|V(S_0)|}.$$

7.4 Summary

Note that the second assumption and the corresponding design parameter N can also be used to limit the search space. The corresponding constraints can be based on individual servers or on sets of servers. In the next section, we will discuss how these constraints can be used in developing mappings.

8 Mapping Algorithms

In this section, we will introduce two algorithms to generate server-to-grid mappings based on the design parameters. The first of these algorithms uses the optimal cluster dimensions to create a mapping. Due to the complexity of this algorithm, we also provide a very efficient second algorithm. The disadvantage of the second algorithm is that, in case independent failures are common, its availability can be low.

8.1 Using Optimal Cluster Dimensions for Obtaining Good Mappings

We have seen above that the optimal dimensions of a server cluster of size $|V(S_0)|$ can be found as

$$x\simeq\sqrt{\frac{k}{l}\times|V(S_0)|}\qquad and\qquad y\simeq\sqrt{\frac{l}{k}\times|V(S_0)|}$$

One can translate this result into an algorithm whose pseudo-code is presented in Figure 6. Intuitively, the algorithm tries to create optimal size clusters and, then, fill the grid with these clusters. Since it promotes grid-locality of virtual servers, intuitively, the resulting grid will have high read availability.

Algorithm:

1. Put the servers into groups such that the servers in each group have the same number of virtual servers.
2. Identify optimal cluster dimensions for each group
3. Try to fill the grid with the given clusters
4. If such a filling is not possible, break some of the clusters into multiple pieces to obtain a final mapping.

Figure 6: Cluster-based mapping algorithm pseudocode

Input:

- A sequence $P =< (s_0, n_0), \ldots, (s_m, n_m) >$ of $(server\_id, number\_of\_virtual\_servers)$ pairs and
- An upperbound on the size of write quorums, k.

Output: A server to grid mapping, $\mu : V \rightarrow G_{l \times k}$.

Algorithm:

1. Sort P in increasing order by n_i and store the result in P_{sorted}
2. Replace each (s_i, n_i) pair with a sequence of $s_{i,1}, s_{i,2}, \ldots s_{i,n_i}$, where each $s_{i,j}$ is a virtual server of s_i. Note that P_{sorted} is now an ordered list of virtual servers grouped by server name.
3. Construct a grid of $\frac{\sum_{i=1}^{m} n(i)}{k}$ rows and k columns.
4. Fill the grid in circular row-major order using successive elements of P_{sorted}:
   (a) Fill rows $1, 3, 5, \ldots$ left-to-right and
   (b) Fill rows $2, 4, 6, \ldots$ right-to-left.

Figure 7: SH-map algorithm

Although this solution would reduce the search space for finding good grid mappings significantly, its computation can still be costly. In the next section, we provide another solution (*snake heuristic*) based on constrained placement of servers to ensure minimal overlapping of the horizontal and vertical extensions of the clusters.

8.2 Algorithm: SH-Map

The snake heuristic mapping algorithm (SH-Map) attempts to balance server load and minimize the maximum overloading experienced any server in the system. The name comes from the way virtual servers are mapped into the grid. The algorithm is depicted in Figure 7. We explain how SH-Map works using an example.

Example 8.1 Let, k, the number of columns in the grid, be 6 and let us be given the following sequence of pairs: $\langle (A, 5), (B, 4), (C, 3), (D, 5), (E, 3), (F, 2), (G, 1), (H, 1) \rangle$. After sorting the sequence and replacing the servers with virtual servers, we get $\langle G_1, H_1, F_1, F_2, C_1, C_2, C_3, E_1, E_2, E_3, B_1, B_2, B_3, B_4, A_1, A_2, A_3, A_4, A_5, D_1, D_2, D_3, D_4, D_5 \rangle$. The final grid is:

$$
\begin{array}{cccccc}
G_1 & H_1 & F_1 & F_2 & C_1 & C_2 \\
B_2 & B_1 & E_3 & E_2 & E_1 & C_3 \\
B_3 & B_4 & A_1 & A_2 & A_3 & A_4 \\
D_5 & D_4 & D_3 & D_2 & D_1 & A_5
\end{array}
$$

$\diamond$

Note that SH-map tends to minimize unwanted intersections between read and write quorums. Therefore, this solution has a good load-balancing property. However, if independent failures are common and if write quorum sizes are relatively small,*this algorithm can suffer from low availability* (in the above example, if server A goes down, 4 out of 6 read quorums are lost). In the next section, we first show that SH-Map algorithm provides close to optimal load balancing, and then use SH-Map to study the properties of the σ and N parameters. Note that SH-Map could still be highly useful in environments with low independent failure probability. Furthermore, it is possible to combine the clustering and SH-mapping algorithms to overcome their shortcomings. Finally, we are currently investigating k-failure quorum systems which can significantly increase the availability. We omit detailed discussions as the focus of this paper is to study the effects of the parameters.

9 Simulations and Experiments

In order to observe the performance of the SH-map algorithm and the effects of the two design parameters we have identified, σ and N, we have designed and conducted a set of simulations and experiments. Simulation software provides a controlled environment where we can carry on various tests. Experiments using real servers, on the other hand, are used for validating that the simulation results indeed reflect the real system performance. In this section, we describe the simulation and experiment architectures, provide results, and discuss their interpretations.

9.1 Experiment Architecture

Our experiment architecture, which models a CDN, consists of a collection of Tomcat and Apache web servers. Apache servers receive requests and pass them onto Tomcat servers that will handle quorum formation and data delivery tasks. We implemented two types of servlet programs, (1) `QuorumServlet` for quorum formation and (2) `ReadServlet` for serving data. A delay value is passed to `ReadServlet` each time it is invoked to enable it simulate the appropriate backend data access delay. Each server starts with a specified capacity, i.e., the number of simultaneous requests is can handle. Client requests are processed as follows:

1. The client selects a server, s, and contacts it as a proxy for read quorum formation. This request is handled by the `QuorumServlet`. Since s can have multiple virtual servers, `QuorumServlet` selects a read quorum using the flat server participation strategy.
2. s then contacts all servers in the corresponding read quorum and gets a timestamp from each one of them. If a server is read-overloaded, then it returns a `busy` signal. In this case, the quorum formation fails and the client is informed of the failure. If none of the servers are overloaded, then the IP address of the server with the most up-to-date copy of the requested object is returned to the client. If more than one server in the read quorum has an up-to-date copy, then one of them is selected randomly based on virtual server representation in the quorum.
3. After receiving the address of the appropriate server, the client sends a data request to the corresponding `ReadServlet`. The `ReadServlet` sleeps for the specified amount to simulate the backend delay. The total amount of time taken to serve the request (including the queuing time) is recorded. Note that if it takes more than a threshold value to serve the request, then the server declares itself read-overloaded and the read fails.
4. For our experiments the number of read operations served by each server is recorded into logs.

We used Java 2.0 for implementation and ran the framework on a set of Pentium III machines running a combination of Windows NT and Windows 2000.

9.2 Using Experiments for Validating the Simulation Environment

We have also used software simulations, based on the equations described in Section 5, to explore various system parameters and mapping strategies. To verify that the simulation environment produces results matching those obtained in a real experimental setup, we ran a series of tests on both frameworks and compared the results.

The simulation environment was configured for grids of dimensions 4×6, 6×4, and 5×5. We randomly created 5 grids for each dimensions, with the following relative server capacity distributions: on the average 40% of the servers have 1, 30% have 2, 10% have 3, 10% have 4, and 10% have 5 virtual servers. The number of servers ranged between 8 and 10. In line with the experimental framework, the flat server participation strategy was used to pick read quorums. For each grid, expected server loads were calculated using the equation

$$L(s_i) = \frac{L}{l} \times \sum_{1 \leq t \leq k} prob(r_t \text{ is chosen for read}) \times \sum_{1 \leq p \leq l} \frac{|V(s_i) \cap V_{p,t}|}{|V_{p,t}|}.$$

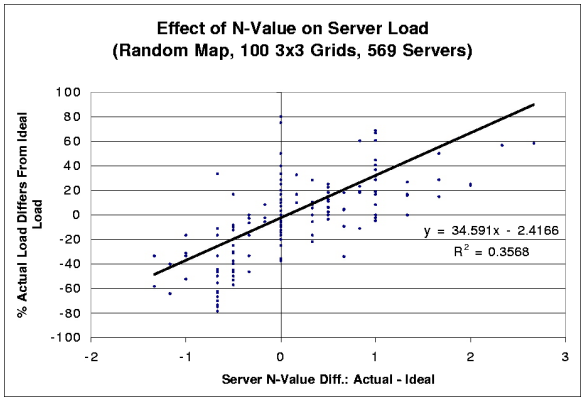
Each grid setup was also run on the experimental framework. The minumum server capacity was 10 simultaneous connections and a backend delay of 10 seconds. We generated 2 requests per second, for a total of 300 requests. Note that none of the servers failed in the experiments, but this load was enough to observe the load distribution.

The cumulative results obtained from the simulation and experimental frameworks were compared in a t-test and an f-test. A t-test compares two datasets (samples) for equal means and returns the confidence level that the sample means are equal. Similar to a t-test, the f-test compares the variances of two samples and returns a confidence value that the variances are equal. Note that we could have used other tests for comparing our results, however, since we have insight into how the experimental environment and simulation operate, we are confident that the t-test and f-test are sufficient. We observed that the confidence levels were 98.9% for the t-test and 73% for the f-test. Furthermore, the corrolation was 98.2%. Thus we are very confident that the simulation results, and hence the equations presented in Section 5, match the real implementation.

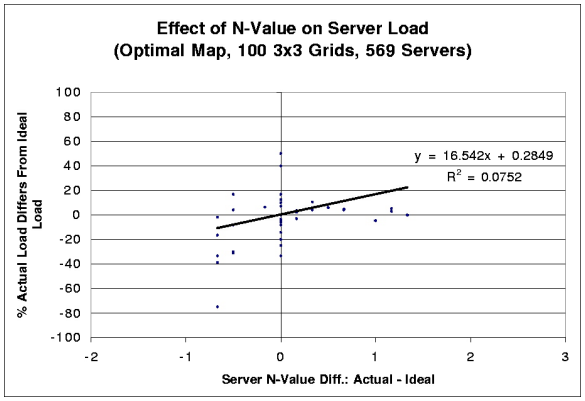
9.3 The Effect of the Parameter N on the Grid Performance

In the second set of experiments, we observed the effect of the second simplifying assumption, the read and write quorum intersections, on the performance of the quorum structure. As discussed earlier, we can capture this assumption, using the parameter N. We ran a set of simulations to observe the correlation between N and the performance of the quorum structure under various mapping strategies. We used large numbers of randomly mapped grids to observe how server load optimality and N are corrolated. In the rest of the paper, we refer to these mappings as the random mapping strategy. We then used SH-mapping to see if (1) SH-mapping provides good results and if (2) it benefits from the correlation between N and the optimality in this process. Figure 8 shows the load and N values for servers in 100 small, 3×3, grids:

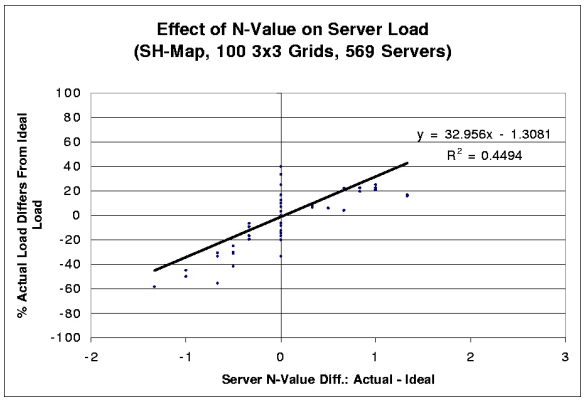
- Figure 8(a) shows the results obtained using the random mapping strategy. Under this strategy, the loads of the servers diverge from their ideal values as large as 100%. Note that Figure 8(b) shows that, under optimal mapping strategies, as expected, the worst case error in the loads of the servers vary less than 50% from their ideal values.
- More importantly, these figures also show that there indeed is a correlation between the load performance of the servers and their N values; as the error in the N values increase, the error in the load also increase. The best-fit



(a)



(b)



(c)

Figure 8:

lines in both figures pass nearly through the origin, illustrating that when a server has the expected N value, then it is also likely to have the expected ideal load.

- Figure 8(c) shows the performance of the SH-mapping: the worst-case error in the load varies between $\pm 40\%$. This is a significant reduction from the worst case (100%) of the random mapping, but the same as the worst-case of the optimal mapping. The average behavior of these two mapping strategies show the same pattern.
- The results also show a parallel trend in the N values, further validating the correlation between N values and the performance of the grid structure. Note that, unlike the optimal and random mapping strategies, which are asymmetric around the origin, the SH-mapping is symmetric, providing an overall better load distribution. Note that the optimal strategy, which does not have such a symmetry, provides the best load distribution. Therefore, such a symmetry is not a necessary condition for optimality.

Although these results show that N is a significant parameter in the design of grid structures, it does not show its effects on larger grid structures. In order to observe these effects as the grid size changes, we have experimented with various grid structures. These grids are filled with servers such that on the average 40% of the servers have 1 , 30% have 2 , 10% have 3, 10% have 4 , and 10% have 5 virtual servers.

Figure 9 shows the observations for two grid dimensions: 10×20 and 20×10. As clearly seen in the figures,

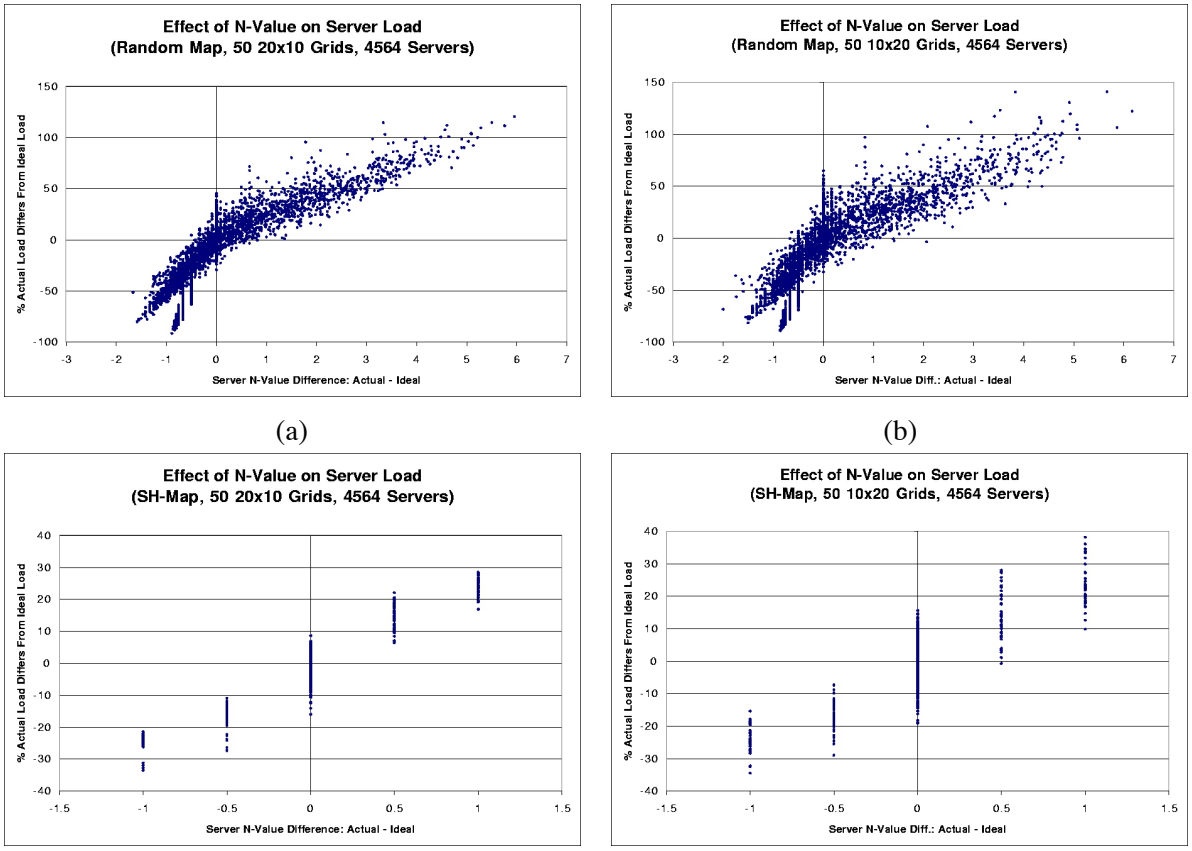


Figure 9:

- in both cases, SH-mapping significantly outperforms random mapping;
- correlations between N and the load performance are clear in all result sets,
- the balancing effect of SH-mapping is visible in both grid dimensions.

Note that the figure also shows that both random and SH-mappings give better results in 20×10 dimensional grids than 10×20 dimensional grids. Although this difference is subtle in random mappings, it is more pronounced on SH-mappings (less than 30% overload in 20×10 grids versus 40% in 10×20 grids). This means that the approach is suitable to CDN applications, where the write quorum size is constrained by the content publication (write) cost.

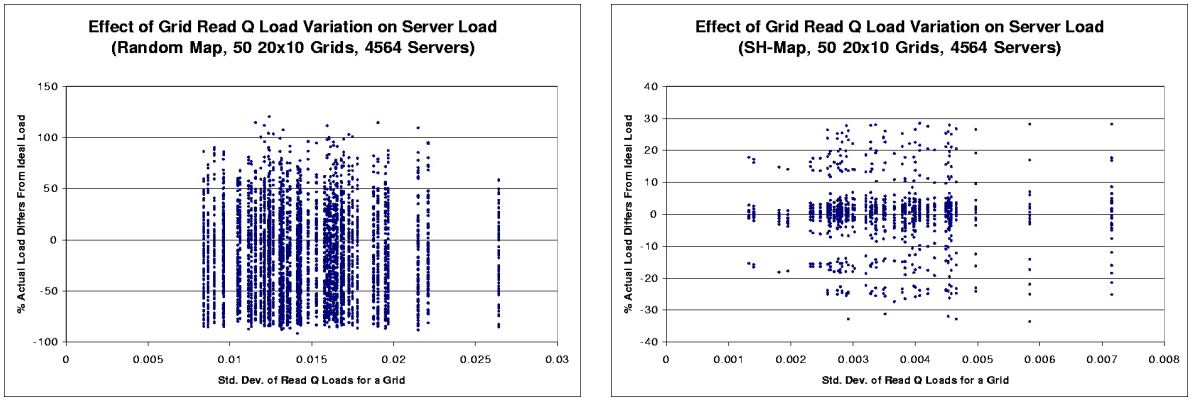
9.4 The Effect of Optimal Cluster Size

In the third set of experiments, we aimed at observing the effect of optimal cluster size. We have created a set of 8 servers, each with two virtual servers, and placed them in various shaped clusters in a 20×20 grid. The rest of the grid is filled with randomly generated servers. The results of this experiment are shown in Figure 10.

The results show that clustering alone helps the servers in the given set. The servers in the cluster were underloaded with respect to the other randomly placed servers in the grid, in all cluster dimensions. Furthermore, as the dimensions of the clusters approach the optimal dimension (in this case 4×4), their load decreases. This means that, given a set of similar servers, their best placement on the grid is in a clustered fashion.

Cluster dimensions	Cluster Underload%
16×1	10.5%
1×16	18.1%
8×2	30.4%
2×8	23.3%
4×4	35.0%

Figure 10: Cluster dimension vs. Percentage underload



(a) (b)

Figure 11:

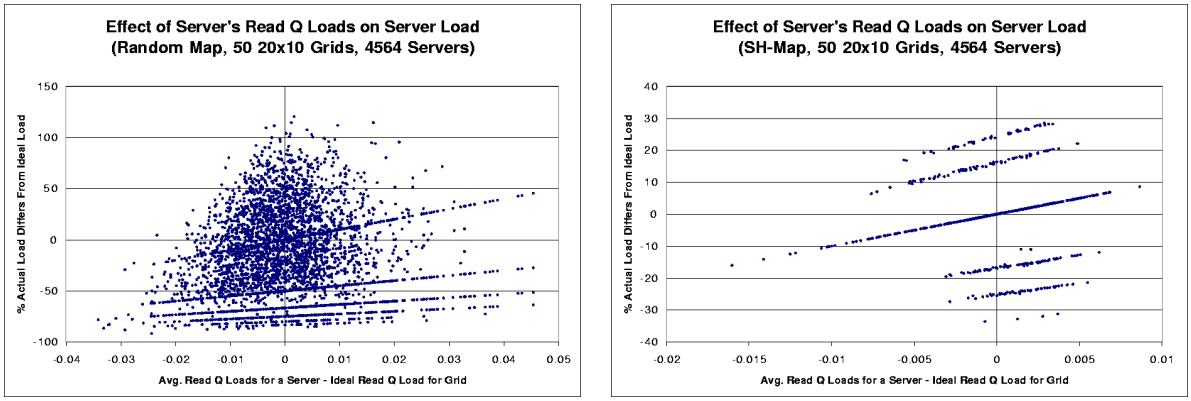
9.5 The Effect of σ on the Grid Performance

In the fourth set of experiments, we observed the effect of the first simplifying assumption, the read quorum load distribution, on the performance of the quorum structure. As discussed earlier, this assumption is captured usign the parameter σ, which states that given k read quorums, each read quorum should have equal, $\frac{1}{k}$, selection probability.

Figure 11(a) shows how server loads are related to the *standard deviation in the read quorum load distribution* in various randomly created mappings. Note that, we can not observe a strong correlation between the server overload and the standard deviation. This means that the worst-case effect of the first simplifying assumption is not large. This is actually very desirable, as it means that we do not need to worry about ensuring that all read quorums have the same load distribution.

We have also observed the relationship between server loads and the standard deviation in the read quorum load distribution in SH-mapped grids (Figure 11(b)). The worst-case overload when SH-mapping is used is around 30% and is significantly lower than when a random mapping is used ($> 100\%$). This is due to even load distribution provided by SH-mapping, as visible in the symmetry of the server loads wih respect to the x axis. Also, please note that, the correlation between the standard deviation value and the server loads is more visible in SH-mapped grids. Although for small standard deviations, the server loads are clustered closer to the x axis, they tend to become more widely distributed for larger standard deviations.

Finally, we ran another set of experiments, where we observed how server loads are related to the σs of the specific read quorums that they are located in. Figures 12(a) and (b) shows the results in random and SH-mapped grids:



(a) (b)

Figure 12:

- First of all, results show that SH-mapping significantly reduces both the deviation in the read quorum loads and the amount overload.
- Secondly, both figures show a correlation between the parameter σ and the server load balance. In the randomly generated mappings, this correlation is overshadowed by the inherent randomness. SH-mapping, however, shows perfect correlation, meaning that it indeed chooses mappings using the first assumption. The reason why there are multiple lines in the graph is that, each N value difference shows itself as a separate line for the parameter σ and that, by its construction, SH-mapping limits the number of possible N value differences.
- Finally, results also show that, as in other observations, SH-mapping induces a symmetry.

9.6 Time Complexity

The time needed to map a set of virtual servers to a grid is of key concern. However, as the number of servers increases and the variation in the number of virtual servers each server has increases, it gets harder to find an optimal mapping. The two algorithms we have used in this paper are exhaustive search and SH-Map. The complexity of each is described below.

The exhaustive search method tries all possible mappings for a grid and returns the mappings which have the lowest maximum overloaded server. It is easy to see that the computational complexity of this method is $O(n!)$ (we have only used it to verify the optimality of other algorithms). On a G4 processor at 450MHz, we need 145 seconds for an average 9-element grid and 1500 seconds (25 minutes) for 10-element grids.

SH-Map, on the contrary, has a low computational complexity and works very fast. The basic operation of SH-Map is to sort the set of servers and the running time is $O(|S|log|S|)$. Using the same hardware used for exhaustive search, SH-Map can fill a 2000-cell grid in approximately 0.01 milliseconds (averaged over 100 runs).

10 Related Work in Quorum Systems

Quorum systems have been widely proposed for years as a means of sharing load, enhancing system availability, and providing consistency in distributed file and database systems. In this section we briefly review properties of quorum systems and explain how our quorum scheme compares to others presented in the literature. First we present an

overview of quorum systems. Then we explain the general method in which quorum systems operate. Finally, we discuss how our CDN quorum solution differs from what has been seen in the past. Recall that equations defining quorums and quorum intersection properties were given in Section 1.2.

Quorum systems are roughly divided into two types. The first type of quorum system uses some kind of voting mechanism to determine when a quorum has been established. In voting systems, each node is given a vote with a certain weight. A quorum is established when the votes of the member nodes surpass some threshold value. The threshold value is set such that the quorum properties hold. Examples of voting protocols include weighted voting [14, 30] and dynamic voting [15, 10]. The second type of quorum systems organizes nodes into a logical structure. Quorums are constructed by selecting nodes from the structure based on some property of the structure. Examples of logically structured systems are Crumbling Walls [25, 26], tree-based protocols [2, 3, 17, 24], hierarchical quorum consensus systems [18], hybrid hierarchical systems [20], the triangular lattice [31], diamond quorums [13], grid-based systems [9, 21, 29, 19, 24], and plane-based systems [23, 6].

For most, if not all, quorum systems proposed to date, read and write operations occur in a similar fashion. It is assumed that some sort of locking protocol is used to control access to an object and that some version control mechanism, like timestamps, is used to identify different versions of the same object. Note that quorum protocols usually enforce mutually exclusive access to objects. Typical read and write operations are as follows. When an object is to be read, a read quorum, $r_i \in \mathcal{S}_{\mathcal{R}}$, is selected and a message requesting access to the object is sent to each server in the quorum. If a server is available, it returns a confirmation message *and a copy of the object*. When all servers in the quorum have responded positively, the requestor takes for its result the most recent version of the object returned. If one or more servers fails to respond to the request (the operation times out) or responds negatively (the object is locked by another operation), the requestor cancels the read, selects another quorum, and tries again. When an object is to be written, a write quorum, $w_j \in \mathcal{S}_{\mathcal{W}}$, is selected and the update is attempted on *all* servers in w_j. The data to be written accompanies the write request. As for reads, if a server does not respond or responds negatively to a write request, the write is cancelled and rolled back if needed, another write quorum selected, and the write is attempted again.

The media object replication scheme proposed in this paper is based on a basic version of the grid-based protocols. Since the application we are addressing is CDN-driven object replication, with potentially heterogeous servers, we can ignore and must change some conditions that are typically found in quorum systems, such as

- In a CDN all updates (writes) are performed by a single writer and the mutual exclusion property does not need to hold for most applications. However, generic quorum systems attempt to enforce mutual exclusion while allowing for multiple writers. Thus Write/Write intersection property does not need to hold for quorum-based object replication in CDNs.
- Quorum operations usually assume that readers have full knowledge of available read quorums. Since readers are not part of the CDN system and should not worry about how CDN works, we can not have such an assumption. Thus, in Constraint III, propose way to make a quorum system work when the readers (the clients) are ignorant of the inner workings of the system.
- The systems referenced above usually assume that servers fail independently of their load. Moreover failures are due to either to network partitions or to server hardware failures. As we mentioned in Section 1.3 this assumption is contrary to actual system behavior.

11 Conclusions

In this paper, we provided an overview of the new challenges associated with replicating large media object over the web. We have highlighted the fact that the replication cost for reflecting the update onto a large number of replica servers may be prohibitively large. We have also discussed why in such systems the traditional assumption of failure independence does not hold and that read and write failures are independent of each other. Given these properties of the Web content replication task, we identified constraints and design parameters to ensure good content replication. We showed the effects of the parameters using theoretical, experimental, and simulation results.

References

[1] D. Agrawal and A. E. Abbadi. Reducing storage for quorum consensus algorithms. In *Fourteenth International Conference on Very Large Data Bases*, pages 419–430, 1988.

[2] D. Agrawal and A. E. Abbadi. The tree quorum protocol: An efficient approach for managing replicated data. In *Proceedings of the 16th VLDB Conference*, pages 243–254, 1990.

[3] D. Agrawal and A. E. Abbadi. The generalized tree quorum protocol: An efficient approach for managing replicated data. In *ACM Transactions on Database Systems, Vol. 17, No. 4*, pages 689–717, 1992.

[4] Akamai Technologies. http://www.akamai.com.

[5] M. K. anf M. Dahlin. Coordinated Placement and Replacement for Large-Scale Distributed Caches. In *IEEE Workshop on Internet Applications*, pages 62–71, 1999.

[6] R. Bazzi. Planar quorums. In *Proceedings of the 10th International Workshop on Distributed Algorithms*, pages 251–268, 1996.

[7] K. S. Candan and W.-S. Li. Integration of database and internet technologies for scalable end-to-end e-commerce systems. *invited chapter submission to Architectural Issues of Web-Enabled Electronic Business*, 2001.

[8] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal. Enabling Dynamic Content Caching for Database-Driven Web Sites. In *Proceedings of the 2001 ACM SIGMOD* , Santa Barbara, CA, USA, May 2001. ACM.

[9] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. In *Transactions on Knowledge and Data Engineering, Vol. 4, Num. 6*, pages 582–592, 1992.

[10] D. Davcev and W. A. Burkhard. Consistency and recovery control for replicated files. In *Proceedings of the 10th ACM Symposium on Operating System Principles*, pages 87–96, 1985.

[11] Digital Island, Ltd. http://www.digitalisland.com/.

[12] Z. Fei. A novel approach to managing consistency in content distribution networks. In *Proceedings of the 6th International Workshop on Web Caching Caching and Content Distribution*, 2001.

[13] A. W.-C. Fu, Y. S. Wong, and M. H. Wong. Diamond quorum consensus for high capacity and efficiency in a replicated database system. In *Distributed and Parallel Databases, Vol. 8, No. 4*, pages 471–492, 2000.

[14] D. Gifford. Weighted voting for replicated data. In *Proceedings of the Seventh ACM SIGOPS Symposium on Operating Systems Principles*, pages 150–159, 1979.

[15] S. Jajodia and D. Muthcler. Dynamic voting algorithms for maintaining the consistency of a replicated database. In *ACM Transactions on Database Systems, Vol 15, No. 2*, pages 230–280, 1990.

[16] J. Kangasharju, J. Roberts, and K. W. Ross. Object replication strategies in content distribution networks. In *Proceedings of the 6th International Workshop on Web Caching and Content Distribution*, 2001.

[17] H. Koch. An efficient replication protocol exploiting logical tree structures. In *23rd Int. Conf. on Fault-Tolerant Computing (FTCS-23)*, pages 382–391, 1993.

[18] A. Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. In *IEEE Tranactions on Computers, Vol. 40, No. 9*, pages 996–1004, 1991.

[19] A. Kumar. An efficient supergrid protocol for high availability and load balancing. In *IEEE Transactions on Computers, Vol. 49, No. 10*, pages 1126–1133, 2000.

[20] A. Kumar and S. Y. Cheung. A high availability sqrt(n) hierarchical grid algorithm for replicated data. In *Information Processing Letters, No. 40(6)*, pages 311–316, 1991.

[21] A. Kumar, M. Rabinovich, and R. K. Sinha. A performance study of general grid structures for replicated data. In *International Conference on Distributed Computing Systems*, pages 178–185, 1993.

[22] W.-S. Li, K. S. Candan, W.-P. Hsiung, O. Po, D. Agrawal, Q. Luo, W.-K. W. Huang, Y. Akça, and C. Yilmaz. CachePortal: Technology for accelerating database-driven e-commerce web sites. In *Demo. at the Very Large Data Bases (VLDB) Conference*, pages 699–700, 2001.

[23] M. Maekawa. A sqrt(n) algorithm for mutual exclusion in decentralized systems. In *ACM Trans. Comput. Syst. 3, 2*, pages 145–159, 1985.

[24] M. Naor and A. Wool. The load capacity and availability of quorum systems. In *SIAM Journal on Computing, 27(2)*, pages 423–447, 1998.

[25] D. Peleg and A. Wool. Crumbling walls: A class of practical and efficient quorum systems (extended abstract). In *Proceedings of the 14th ACM Symposium on the Principles of Distributed Computing (PODC 95)*, pages 120–129, 1995.

[26] D. Peleg and A. Wool. The availability of crumbling wall quorum systems. In *Discrete Applied Math, 74(1)*, pages 69–83, 1997.

[27] P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed internet replica placement. In *Proceedings of the 6th International Workshop on Web Caching and Content Distribution*, 2001.

[28] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond Hierarchies: Design Considerations for Distribued Caching on the Internet. In *ICDCS*, 1999.

[29] O. E. Theel and H. Pagnia-Koch. General design of grid-based data replication schemes using graphs and a few rules. In *International Conference on Distributed Computing Systems*, pages 395–403, 1995.

[30] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. In *ACM Transactions on Database Systems, Vol. 4, No. 2*, pages 190–209, June 1979.

[31] C. Wu and G. G. Belford. The triangular lattice protocol: A highly fault tolerant protocol for replicated data. In *Proceedings of the 11th IEEE Symposium on Reliable Data Systems*, pages 66–73, 1992.

[32] Zona Research. http://www.zonaresearch.com/.