Server Replication in Interactive, Push-Based Data Delivery Networks

Nikhil Iyer
Computer Science and Engineering Department
Arizona State University, Tempe, AZ 85281, USA
Email: nikhil.iyer@asu.edu

K. Selçuk Candan
Computer Science and Engineering Department
Arizona State University, Tempe, AZ 85281, USA
Email: candan@asu.edu

***Abstract*— In a push-based system, updates that are made to objects are sent to clients, without them explicitly requesting these updates. The focus of this research is to develop a server replication protocol for such interactive, push-based networks. When a server gets overwhelmed and is serving more clients than it can handle, a replica is created. After the replication, a portion of the clients connected to the overwhelmed server are redirected to the new replica. The decisions (a) "at which particular server the replica is created" and (b) "which clients are redirected" can greatly influence the performance observed by the clients, the level of the service provided by the server, and the overall level of satisfaction with the functioning of the servers. In this paper, we identify a number of parameters that influence the selection of a new server and we develop an efficient and effective protocol, based on these parameters. The performance of the protocol is investigated through simulation. The protocol performs close to an ideal placement of objects and clients. On average a savings of 25% in was observed when compared with an uninformed strategy.**

I. Introduction

Web sites can serve static or dynamic data; but conventional web servers (such as the popular Apache Web server) do not allow users to interact with each other; i.e, a user/client who enters a web site is unaware of the presence of other clients visiting the same site. There is no interaction between these clients. Recently a number of companies, such as Adobe Atmosphere [17], made available web servers capable of serving 3D virtual worlds where users can interact with each other as well as the objects in the world (Figure 1). Therefore the content provided by these servers are not only dynamic and visually rich, but also interactive [7], [8], [11].

In an interactive distributed client-server system, each client views, updates, and/or tracks changes made to a set of objects. In such a system, any update to any object has to be sent to all other clients interested in this particular object (Figure 2). Therefore, deployment of Internet-scale 3D worlds poses many technical challenges. First of all, the network can cause variable access latency and this may introduce consistency challenges. The data consistency challenge in such distributed systems has been dealt by [5], [6]. In addition, rapidly changing 3D content requires large bandwidth and puts heavy loads on the 3D servers. From time to time, as the number of clients interested in a particular object increases, it is possible that the server serving this object might get overwhelmed. When this happens and when the quality of service falls below a level that is acceptable, steps need to be taken to remedy the situation.

One widely used approach dealing with server overload is the creation of replicas of the existing object and the server. The focus of this research is on identifying a replica server from a selection of servers in such a way that the benefit to the client is maximized and the network cost associated with maintaining the system is kept as low as possible. Although considerable work has been done in the field of server replication [9], [10], [4], [15], [14] a vast majority of this work is in the field of pull-based systems. Our goal, on the other hand, is to develop a protocol/algorithm that addresses the unique challenges introduced by interactive, push-based systems.

Conventional data management systems are pull-based: queries or transactions are executed only when they are explicitly requested by a user or an application program; i.e., transfer of data from servers to clients is initiated by an explicit client pull (Figure 2). However in a push-based (or a publish/subscribe) system, updates are sent to clients not when they request it, but as needed (for instance, depending on the rate of updates to a particular object). In an environment where multiple users are interacting within a virtual world, an update to the spatial location of an object (say, avatar) has to be transmitted to all the clients that are in its vicinity.

When a server is overwhelmed, an available network node needs to be selected to replicate the objects present on the overwhelmed server. To reduce the load, a portion of the clients currently connected to the server being overwhelmed need to be redirected to the new server. In this paper, we propose a server replication scheme that maximizes the benefit to the end-users while minimizing

Fig. 1. An interactive Atmosphere environment (picture taken from http://virtualuniverse.dyndns.org)

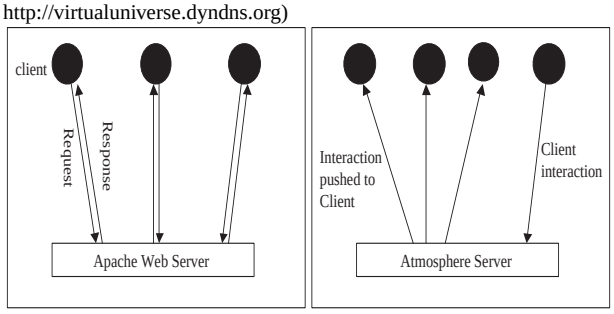

Fig. 2. Pull-based content delivery versus interactive, push-based delivery

the network cost of the system. We study the factors that contribute to the quality observed by end-users and the network cost. We use these factors as parameters while making decisions regarding the replication of the server and selection of clients to be redirected. The goal of our work is to achieve these tasks with as much local information as possible. The protocol proposed in this research is designed to be used as part of a *push-based content delivery network* with the ability to easily and transparently spawn new copies of the servers as and when the requirement arises.

A. Related Work

In the Internet, it is often advantageous to place content as close to users as possible in order to remove sources of network delay [21], [22], [23], [24], [25]. Server farms, provided by various companies including Digital Island [26], MirrorImage [28], are possible external scalability solutions. Several high-technology companies are competing feverishly with each other to establish network infrastructures refereed to as a *content delivery networks (CDNs)* [27]. 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. Traditionally, CDN replication has been treated as a facility placement problem [21], [24].

Liu, et al. describe some of the salient features that distinguish push and pull based systems [3]. PASTRY [1] provides efficient and fault-tolerant routing, object location and load balancing within a self-organizing overlay network. SCRIBE[2], built on top of PASTRY[1] is a publish/subscribe system. Rendezvous point forwards the data along the multicast tree formed by the reverse paths from the rendezvous point to all subscribers. Fei et al. propose a scheme to allocate servers to clients in a way that minimizes clients' response times in multicast systems [13]. Crovella et al. study techniques to find good service providers without prior knowledge about the server location and network topology. They consider the use of two distance metrics in the Internet: hops and round-trip latency [4]. Xu et al. propose a data replica placement strategy and the concept of transparent replica proxy replication [14]. They define a cost value which is a function of read rate, write rate and distance between the clients and servers. [18] introduces the Constrained Mirror Placement (CMP) problem to reduce round trip time between clients and mirrors and the load on the servers. Sivasumramanian et al. propose dynamic selection of replication and caching strategies. The choice of the best strategy for a particular situation is made by simulating the recent past and observing the performances of different strategies[19]. Pang, et al. propose object placement strategies [20] for distributed multiplayer games.

Streaming data (such as live video) services on the Internet is another example of a push-based system. Once a client has subscribed to a streaming service, streaming data (such as a live video) is sent to the client without the client making an explicit request. The problem with providing commercial quality streaming services on the Internet are scalability (the possibility of a large user population being physically wide spread makes scalability an issue), QoS concerns (clients have QoS expectations), reliability, and Intelligent server replication is also used to overcome the QoS concerns and reliability [15] for streaming data systems. Recently there has been considerable interest in using proxies and application level overlays for streaming data delivery [29], [30], [31], [32], [33]. In addition to these mentioned research activities, many CDN vendors provide various solutions for streaming media delivery on the Internet. In the solutions provided by Akamai[27], streaming media is distributed close to the users. It also reduces bandwidth requirement at the origin media server site because media is served from the edge cache servers. RealProxy by RealNetworks[34] is software installed on a network or ISP gateway that aggregates and handles client requests for media streamed. Replication improves reliability and availability [35], [36], [37]. The main difference between server replication schemes for video delivery and the schemes we propose in this paper is that, in most existing systems, data originated in content providers' data servers.

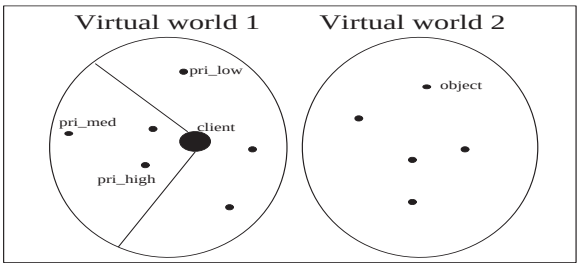

Fig. 3. Two virtual worlds, and the objects within, served by the same server. A client is viewing one of these virtual worlds; i.e., the objects in this world is in the hotset of the client. The objects are assigned priorities based on how important they are for proper visualization of the world: closer objects are given higher priorities.

In the architecture we investigate in this paper, however, updates to the objects are initated by the end-users of the system themselves and have to be propogated to the other end-users interested in seeing the updates.

B. Contributions of this Paper

Our focus in this paper investigating server replication schemes for distributed systems with the following properties:

- *push-based:* each end-user has a set of objects of interest (and this set can change over time) and changes to objects in this set needs to be propogated to the end-user continuously.
- *interactive:* updates to the objects are very frequent and (instead of being driven by the content-provider) they are initated by the end-users of the system and have to be propogated to the other end-users interested in seeing the updates),

In this paper, we identify a number of parameters that influence the selection of replica servers and we develop an efficient and effective protocol, based on these parameters.

The rest of this paper is organized as follows. Section II presents a formal specification of the problem domain. Various relevant parameters are introduced, the properties of the system and are highlighted, and an update propagation cost function is developed. Section III explains the design and implementation of the proposed protocol. Section IV presents experimental results for evaluation and analysis of the protocol. The experimental setup for the above mentioned experiments is also described. Section V concludes this paper.

II. Interactive Push-Based Content Delivery: Problem Overview

We model the network as a connected graph *G(V,E)*, where *V* represents the set of clients (*C*) and servers (*S*), and *E* represents the network connections between clients and servers. *G'(V,E')*, the reachability graph of *G* is a clique, where *E'* represents the edges in the reachability graph *G'* and each edge corresponds to the set of shortest paths between nodes in *V*. Note that each server may not have a full, up-to-date copy of this reachability graph.

An object represents any kind of interactive data resource. It could be an application, a file, or an avatar in a virtual world. Each object has a source/home, $source(o_k)$ where $o_k \in O$. The source of an object is the server node that has the original copy of the object. This can be compared to a rendezvous peer [1] in a multicast tree or a origin source in CDNs.

Each client node subscribes to one server node for a single object. A client has a number of objects that it may want to view. This set of objects is called the clients' hotset, *hotset(c_i)*, where $c_i \in C$. Each client associates a *priority* with each object. The priority represents how important the object is to the client (Figure 3).

Updates to objects are initiated by clients. A client may update the state of any of the objects in its hotset and has to see any changes to its state made by other clients. These updates are transmitted in a push-based manner to all the other clients viewing/interacting with this object.

Each server hosts at least one object and often more. Each server has a *capacity* associated with it. The value of capacity collectively represents constraints imposed on the server, in terms of memory, local bandwidth, and processing capabilities. If the load on the server exceeds the server's capacity, then it has to offload some of the services to an other server. The following steps ensure consistency among the replicated servers:

1) A client c_i updates (changes the state) an object o_k present on one of the replicated servers s_j (the server the client is connected to).
2) The update is transmitted from server s_j to the server which is the source of the object o_k.
3) The update is then transmitted from the source of the object to all the servers that have a copy of the object o_k.
4) Each server then transmits the update to all client that are currently connected to it and are interested in viewing updates made to the object o_k.

A. Content Replication and Client Redirection

When a server is overwhelmed, a new server node needs to be selected to replicate the objects present on the overwhelmed server. Clients currently connected to this server need to be redirected to the new server. The entire process of server replication consists of the following steps

1) An overwhelmed server makes the decision to replicate.
2) A replica of the server is created at a suitable network node.

3) A portion of the clients connected to the replicated server are redirected to the newly created server.

In this paper our focus is to determine which network node to use for replication when a server gets overwhelmed and which clients to redirect to the new server. The main criterion we use for making these decisions is the maximization of end-user satisfaction and minimization of the update propagation cost.

In this research, we focus on creating a single copy of a server, when it gets overwhelmed. Obviously, there are advantages of creating multiple replicas simultaneously when the network receives a surge of new clients. Spawning multiple replicas can help tackle such surge conditions. The proposed protocol can be easily extended to create more than one replica simultaneously.

If the decision to replicate is triggered after a server gets overwhelmed, there is the danger of the server not being able to provide service to the existing clients until a new server is created and some clients are redirected to the new server. In order to avoid such a situation, some kind of forecasting technique is required to predict future client demand and create new servers even before a server get overwhelmed, ensuring continuous service. In this paper, we do not tackle the issue of deciding when to trigger server redistribution and assume that a separate process handles the decision of when to trigger replications.

In the next subsection, we introduce the various parameters used in the rest of the paper.

B. The Constraints that the System has to Obey

In this subsection, we model the salient characteristics of the push-based content delivery system. Table I presents the list of parameters that are used in this section. Clients subscribe to objects in their *hotsets*:

- a client c_i subscribes to a server s_j for an object o_k only if o_k is in the clients *hotset*,
- if the object o_k is in the clients c_i *hotset*, then c_i subscribes to only one server for o_k.

Therefore,

$$\forall i \forall k \sum_j Y_{i,k,j} = Z_{i,k} \tag{1}$$

The system ensures objects availability: each object o_k is placed on at least one server s_j:

$$\forall k \sum_j X_{k,j} \geq 1 \tag{2}$$

A client c_i can subscribe to a server s_j for an object o_k, only if s_j has the object o_k:

$$\forall i \forall k \forall j (Y_{i,k,j} \leq X_{k,j}) \tag{3}$$

Symbol	Meaning
C	the set of clients. $c_i \in C$ denotes a client.
S	the set of servers. $s_j \in S$ denotes a server.
O	the set of objects. $o_k \in O$ denotes an object.
$pri_{i,k}$	the priority client c_i associates with object o_k.
$freq_{i,k}$	the expected frequency with which client c_i updates object o_k.
source(o_k)	the source/home of object o_k. This is similar to the rendezvous peer in [1]. An update made to o_k is transmitted to *source*(o_k), and from there to all servers that have a copy of o_k.
size(o_k)	the size of object o_k. Size abstracts to the amount of resources required to maintain the object on the server.
cap(s_j)	the capacity of server s_j. This denotes the total amount of resources available on the server for objects.
$D_{i,j}$	the delay between network node *i* and network node *j*.
$R_{i,j}$	the resources needed to push an update from network node *i* to network node *j*.
T_j	the maximum number of clients a server s_j can provide service to.
$X_{k,j}$	whether object o_k is present on server s_j.
$Y_{i,k,j}$	whether client c_i subscribes to server s_j to view object o_k.
$Z_{i,k}$	whether object o_k is in client c_i's *hotset* (*hotset*: a object is in a client's *hotset* if the client is interested in viewing and/or making changes to the object).
$A_{i,j}$	whether client c_i is attached to (subscribed to at least one object on) server s_j.

TABLE I

SYSTEM PARAMETERS

The sum of the sizes of the objects on a server s_j does not exceed the maximum capacity of the server:

$$\forall j \sum_k (size(o_k) \times X_{k,j}) \leq cap(s_j) \tag{4}$$

The number of clients that a server s_j can service is less than or equal to the threshold:

$$\forall j \sum_i A_{i,j} \leq T_j \tag{5}$$

Given these constraints, (explained next) the update propagation cost needs to be minimized and user satisfaction is maximized.

C. Objective: Maximization of end-user Satisfaction and Minimization of Update Propagation Cost

The goal of proposed replication protocol is to determine where to create a replica of an overwhelmed server and which clients to redirect to the new replica. There is a certain cost associated with maintaining a replicated system: updates have to be propagated to the clients interested in these updates.

1) Network Cost: The effect of updates made by a client c_i on an object o_k can be computed as follows (Figure 4): Client c_i is subscribed to server s_j and *source*(o_k) is the source of the object updated by c_i. Each update is sent from c_i to s_j and from s_j to *source*(o_k).

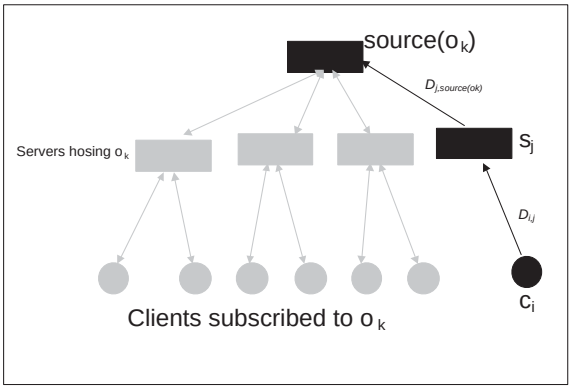

Fig. 4. Client c_i updating object o_k on server s_j

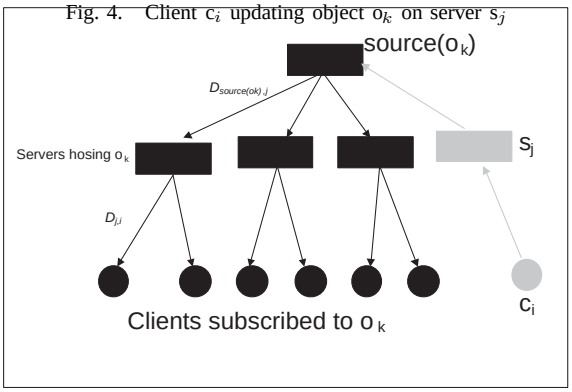

Fig. 5. Update to object o_k being pushed to all interested clients

Therefore, the update cost of this user for all updates in a unit time can be computed as

$$freq_{i,k} \times (R_{i,j} + R_{j,source(o_k)}), \tag{6}$$

where $freq_{i,k}$ is the frequency with which c_i updates object o_k. Each update to o_k is, then, sent from *source(o_k)* to all servers that have a copy of o_k and from those servers to all clients interested in viewing o_k (Figure 5). The cost of pushing updates to an object is the product of the *total frequency* with which the object is updated and the cost associated with pushing a single update to the object. Therefore, the network cost of pushing updates to interested clients is

$$\sum_i freq_{total(k)} \times \sum_j (R_{source(o_k),j} + R_{j,i}) \tag{7}$$

The system should minimize the overall network cost:

$$net\_cost = \sum_{i,k,j} freq_{i,k} \times (R_{i,j} + R_{j,source(o_k)}) \times Y_{i,k,j} + \sum_{i,k,j} freq_{total(k)} \times (R_{source(o_k),j} + R_{j,i}) \times Y_{i,k,j}$$

2) User Dissatisfaction due to Delays: Each update to object o_k travels from the updating client, c_u, to an observing client, c_i, through server $source(o_k)$. The end-to-end delay can therefore be computed as

$$(D_{u,j} + D_{j,source(o_k)}) + (D_{source(o_k),l} + D_{l,i}) \tag{8}$$

If an object is updated more often then the others, then this delay is will be more visible and bothersome to the end-users:

$$\sum_u freq_{u,k} \times \Big((D_{u,j} + D_{j,source(o_k)}) + (D_{source(o_k),l} + D_{l,i})\Big) \tag{9}$$

Furthermore, the priority each client assigns to objects also affect the degree of dissatisfaction caused by delays:

$$pri_{i,k} \times \sum_u freq_{u,k} \times \Big((D_{u,j} + D_{j,source(o_k)}) + (D_{source(o_k),l} + D_{l,i})\Big) \tag{10}$$

Therefore, the effects of updates to an object on the client satisfaction is the product of the client's priority, the frequency with which the that particular object is updated, and the delays associated with the updates.

The proposed system aims to minimize the worst-case user dissatisfaction with the system:

$$user\_dis = \max_i \sum_k pri_{i,k} \times \sum_u freq_{u,k} \times ((D_{u,j} + D_{j,source(o_k)}) \times Y_{u,k,j} + (D_{source(o_k),l} + D_{l,i}) \times Y_{i,k,l})$$

Our goal is to minimize the negative effects of the updates on the end users while also keeping the network overhead low.

III. Achieving the Objectives

The parameters that can affect the user satisfaction and the network cost are both user and system dependent. Before we propose a solution, we first discuss the reasons why one may prefer to choose a particular server (s_x) for replication over the others when a server (s_j) is overwhelmed.

- *Observation I:* If a large number of client nodes are subscribed to s_j from the same geographical location close to s_x and far from s_j, then placing a replica of the object at s_x may reduce the overall update propagation cost.
- *Observation II:* If clients that are subscribed to s_j and that are close to s_x are viewing objects that are updated more frequently than other objects, then this may also reduce the overall update propagation cost. If a replica of the frequently updated object is placed

at s_x, then the total number of update messages will be reduced (instead of sending an update to every client, just one is sent from the source of the object to s_x and then from s_x to the clients).
- *Observation III:* If clients that have high priorities for a particular object located at s_j are close to s_x, then the delay associated with the viewing/making changes to the object need to be kept at a minimum. If a replica of the object is located closer to such clients, then the overall client satisfaction is maximized.
- *Observation IV:* If s_x has large amount of residual capacity (space, processing power, bandwidth), then it may be a good candidate for replication. This means that s_x is currently not overburdened and placing a replica there is not likely to overwhelm s_x.

The goal of this research is to develop a distributed protocol/algorithm that achieves the objectives stated in II-C. The protocol proposed in this research aims replacing a single large server by a *push-based content delivery network*, with the ability to easily and transparently spawn new copies of the servers as and when the requirement arises (*growing phase*) and shrink the overall delivery network when the load drops (*shrinking phase*). We use the parameters described in Table I to determine where and when to create server replicas. All these parameters influence the performance of the system and, therefore, should be used when selecting a new replica server. In the following sections, we present the growing and shrinking protocol used for managing the push-based content delivery network.

A. Overview of the Growing Protocol

In this section, we present the overview of the protocol used for determining which server to use for replication when a server is overwhelmed. Our primary goal is to ensure that the protocol can be implemented in a distributed way: each server should be able to choose how to grow based on the information available to itself, without having to retort to the use of a central authority.

Let *num_clients* be the number of clients subscribed (viewing one or many objects) to the current, overwhelmed server. The server that becomes overwhelmed uses the following high-level steps to choose which server will contain a new replica of the objects it contains:

1) The overwhelmed server should choose one (or more) servers, among the ones it *knows* about, for replication. In this paper, we do not address the issue of deciding when to trigger server redistribution. We assume that there is a separate process at the server which identifies when to trigger redistribution based on the local resources as well as based on the growth rate.
2) The overwhelmed server should choose a portion of the clients it is serving for *redirection* to the new server(s).

The following subsections provides details of the proposed protocol.

1) Selecting Candidate Servers: Let s_j be the overwhelmed server. Among the servers that s_j knows about, those that have enough capacity for replication and serving new clients are chosen as candidates:

- The server must have residual capacity greater than the size of the server being replicated.
- The server must have the ability to host $\frac{num\_clients}{num\_of\_new\_replicas}$ more clients. In this research, we focus on creating a single copy of a server and do not tackle simultaneous creation of multiple replicas when a server gets overwhelmed.

2) Choosing the Right Candidate Server: The overall objective is to minimize the update and push costs for all clients, objects, and replicas as the demand grows. In Section II, we have seen that this involves a number of parameters including (a) objects locations, sizes, and update rates (b) server locations and capacities, (c) network topology and the delays between servers, and (d) clients needs, hotsets, and object priorities. Therefore, an optimal decision as to where to replicate would require global information, such as the *total update rate to objects being replicated* and the *maximum priorities clients attach to these objects*.

In order to accommodate both priority and update frequency of the objects, without having to collect global information, we introduce the concept of *criticality*, which is defined as the product of the priority a given client has for a particular object and the frequency of this client's update rate for this particular object. Criticality of an object o_k for a client c_i is defined as follows:

$$CT_{i,k} = freq_{i,k} * pri_{i,k} \tag{11}$$

Intuitively, this represents how important a particular object is to a client. Criticality can be computed for an object/client pair without any global information.

As discussed in Section II, for each object o_k viewed by a client c_i on the server s_x, there is an associated update propagation cost:

$$dist\_cost_{i,k,x} = D_{i,x} + D_{x,source(o_k)} \tag{12}$$

Therefore, for each candidate server, s_j can calculate a *server_score* that captures (a) the needs (*criticality*) of the local clients and (b) the update propagation cost for the

local objects ($dist\_cost$):

$$server\_score(s_x) = \sum_i \sum_k CT_{i,k} * dist\_cost_{i,k,x} * Y_{i,k,j} \quad (13)$$

The server s_j will select the candidate server s_x with the lowest $server\_score$ for replication. Note that s_j calculate the value of $server\_score(s_x)$ without knowing which clients will be the redirected to the selected replica server. Therefore, the next task of s_j is to choose the clients to redirect.

3) Selecting Clients to Redirect: As we stated in the previous section, in this paper we do not consider simultaneous creation of multiple replicas when a server gets overwhelmed. Once the appropriate server for replication has been determined, a portion of the clients connected to the original server need to be redirected to the newly created replica.

For each client c_i connected to itself, the overwhelmed server s_j calculates the added cost associated with redirection. Let client c_i be viewing object o_k through server s_j and let s_x be the newly created replica of s_j. Then, s_j can compute a penalty value associated with redirection of this client:

$$addedcost_i = \sum_k CT_{i,k} * (dist\_cost_{i,k,x} - dist\_cost_{i,k,j}) \quad (14)$$

Clearly, s_j would like to select the clients with the lowest penalty for redirection. Redirecting the lowest-penalty $\frac{num\_clients}{2}$ clients to the new server ensures that clients are redirected to the new server in such a way that the total increase in cost of maintaining (clients updating objects and those updates being transmitted to interested clients) the system is minimum.

4) Overhead Analysis: The server may need to exchange information collection messages with the clients connected to it. These are all local information collection messages and can regularly be refreshed so that they do not need to be collected after the server faces overload. In either case, these steps are performed once for each replication and are amortized over time through savings in update propagation.

B. Overview of the Shrinkage Protocol

If a situation arises in which a server is being consistently underutilized, then steps need to be taken to avoid a prolonged waste in resources. This can be achieved by directing the clients connected to the underutilized server to another existing server. This phase of the protocol is called the shrinkage phase. The shrinkage protocol should achieve the following goals.

- Eliminate unnecessary replication.
- Avoid repeated growth followed by shrinkage of the resources.
- The clients are redirected to a new server in such a way that it does not cause the new server to get overwhelmed.

When a server s_j is consistently underutilized, we use a protocol similar to the growth protocol. For all candidate servers, s_j computes server scores. Then, *all* clients connected to s_j are redirected to the server s_x, with the lowest server score. If s_j is the source of any object, s_x is declared to be the new source of the object.

Note that the shrinkage phase does not create a new server, but uses an existing server. the growth algorithm, however, did not assign an added advantage to a server that already has a copy of the objects present on the replica that is being *eliminated*. If a server has a copy of the object already, then there is no need to replicate that object at the new server. Let s_j be the server that is to be shrunk. When deciding on which server, s_x, to shrink to both $server\_score(s_x)$ and the number of objects that are present on both s_j and s_x are considered.

IV. Experimental Evaluation

In this section, we present simulation-based experiments that have been conducted for various network configurations and various other system parameters.

A. Experiment Setup

In this section, we compare the results obtained by the (a) *proposed protocol* have been compared with the results obtained by

(b) *random* placement, where

- client goes to the server closest to it to access the object and
- if a server gets overwhelmed, a new server is randomly chosen for replication;

(c) *user ideal* placement, where

- the client placement and replication strategy are determined by solving non-linear mixed integer problems specified based on the constraints stated in Section II subject to the *least possible user dissatisfaction* objective function. We used LINGO to generate ideal solutions.

(d) *network ideal* placement, where

- the client placement and replication strategy are determined by solving non-linear mixed integer problems specified based on the constraints stated in Section II subject to the *least network cost* objective function. We, again, used LINGO to generate ideal solutions.

(e) *no_priority protocol* based placement, where

- the objects are replicated using the proposed protocol, except that user priorities are ignored.

The *no_priority protocol* based placement is essentially a *heuristic* to the network ideal placement.

	Net.Ideal	Rand.	**Prot.**	Prot. (no pri.)
Average	12745	145	855	815
Highest	43000	637	1242	1135
Lowest	7600	22	572	581

TABLE II

OVERVIEW OF THE TIMING RESULTS (MILLISECONDS)

	Net.Ideal	Rand.	**Prot.**	Prot. (no pri.)
Av.Net.Cost	1.000	1.241	1.043	1.047
Ave.UserDis.	1.000	1.258	0.954	0.989

TABLE III

OVERVIEW OF THE (RELATIVE) OPTIMIZATION RESULTS

B. Overview of the Results

The *user ideal* and *network ideal* placements requires full knowledge about the networks and users. Furthermore, unlike the proposed protocol that could be implemented in a distributed manner, they relied on an expensive non-linear mixed integer problem solver. For instance, for a *very small* network of 5 servers, 5 objects, and 50 clients, while the proposed protocol took only 200ms to compute, the network ideal required 2000ms and the user ideal required 887000ms. For this setup, network ideal returned results within 1.3% of the user optimal and the proposed protocol returned results within 1.4%. A random assignment, however, was more than 25% more costly in terms of user satisfaction. Since computing user ideal assignments are prohibitively expensive, in the rest of the experiments, we use the network ideal as the base of comparison.

The parameters varied during the course of the experiments were the number of clients (50-200), number of servers (5-10), number of objects (5-25) and the network topology (random vs. transit-sub; transit-sub networks have been generated using GT-ITM [12]). The priorities that clients have for objects range from 0-10 and are assigned using Uniform distribution. Hotsets and the frequency of update is assigned randomly. Table II provides details regarding the time each protocol takes to execute. The numbers in the table represent the amount of time it takes a particular algorithm to identify a placement strategy of objects and clients. As expected, the random placement protocol has the least execution time. On the other hand, the proposed algorithm identifies close to placement strategies in a fraction of the time required by a network ideal algorithm. Note that the table does not list the timing for *user ideal* placement; the reason for this is that for the size of problems we have experimented, the *user optimal* placement requires many hours of computation or fails to provide any answers even for relatively small configurations (100 clients, 10 servers, and 5 objects), therefore is not feasible.

Table III shows an overview of the optimization results obtained from the experiments we conducted. This table shows that, when it comes to network cost, the proposed protocol generates networks very close to the optimal (within 4%) without requiring global knowledge about the network, objects, and clients. Furthermore, the resulting user dissatisfaction is smaller than the network ideal placement (around 5%) and significantly smaller (around 25%) than random placement. These indicate that the proposed protocol provides highly effective placement strategies, in a fully distributed manner and without requiring global knowledge about objects and clients.

To see if the proposed protocols is still feasibile for larger networks, we have also experiment with much larger network sizes: for a network with 200 nodes, 100 clients, 20 servers, and 50 objects, the time required by the proposed protocol still stays within 30 seconds. Note also that both *network ideal* and *user ideal* placements require knowledge about the global state of the network, objects, and clients; therefore are not practical in real system deployment.

C. Individual Effects of Various System Parameters

In Tables IV- X, the effect of various parameters (servers, clients, objects and network topology) on the performance of the protocol is shown.

Network Cost (5 objects, 100 clients)				
#Servers	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
5	1.000	1.241	1.043	1.047
10	1.000	1.275	1.040	1.040

TABLE IV

EFFECT OF THE NUMBER OF SERVERS ON NETWORK OPT.

User Dissatisfaction (5 objects, 100 clients)				
#Servers	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
5	1.000	1.30	0.936	0.973
10	1.000	1.26	0.972	1.006

TABLE V

EFFECT OF THE NUMBER OF SERVERS ON USER OPTIMIZATION

Execution time in ms (5 objects, 100 clients)				
#Servers	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
5	7600	22	572	580
10	9750	50	795	750

TABLE VI

EFFECT OF THE NUMBER OF SERVERS ON OPTIMIZATION TIME

1) Servers: As shown in Table IV, as the number of servers in the system increases, the proposed protocol performs slightly better in terms of reducing the network cost. Table V, on the other hand, shows that the effectiveness of the proposed protocol (relative to that

of the network ideal placement) reduces as the number of servers in the system increases. As expected, the execution time of the algorithm increases with the number of known servers (Table VI).

Network Cost (5 objects, 10 servers)				
#Clients	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
50	1.000	1.376	1.079	1.072
100	1.000	1.275	1.040	1.040
200	1.000	1.284	1.090	1.082

TABLE VII

EFFECT OF THE NUMBER OF CLIENTS ON NETWORK OPT.

User Dissatisfaction (5 objects, 10 servers)				
#Clients	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
50	1.000	1.601	1.046	1.123
100	1.000	1.256	0.972	1.006
200	1.000	1.266	1.024	1.084

TABLE VIII

EFFECT OF THE NUMBER OF CLIENTS ON USER OPTIMIZATION

Execution time in ms (5 objects, 10 servers)				
#Clients	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
50	4800	15	632	628
100	9750	50	794	750
200	13250	324	1242	1136

TABLE IX

EFFECT OF THE NUMBER OF CLIENTS ON THE OPTIMIZATION TIME

2) Clients: As shown in Table VII, as the number of clients in the system increases, the proposed protocol performs slightly worse in terms of reducing the network cost. Other hand, in terms of user satisfaction, the effectiveness of the proposed protocol improves as the number of clients in the system increases (Table VIII). As expected, the execution time of the algorithm increases with the number of clients in the system (Table IX).

Network Cost (10 servers, 100 clients)				
#Objects	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
5	1.000	1.275	1.040	1.040
10	1.000	1.282	1.076	1.065
25	1.000	1.335	1.026	1.023

TABLE X

EFFECT OF THE NUMBER OF OBJECTS ON NETWORK OPT.

3) Objects: In Table X, the effect of change in the number of objects on the network cost is shown. As the number of objects in the system increases, the proposed protocol performs slightly better in terms of reducing the network cost. In terms of user satisfaction, as the number of clients in the system increases, the effectiveness of the proposed protocol stays constant relative to the effectivess of the network ideal mapping(Table XI). As expected, the execution time of the algorithm increases with the number of objects (Table **??**), but the increase is much slower than that of network ideal solution.

User Dissatisfaction (10 servers, 100 clients)				
#Objects	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
5	1.000	1.256	0.972	1.006
10	1.000	1.259	1.009	1.053
25	1.000	1.407	0.998	1.049

TABLE XI

EFFECT OF THE NUMBER OF OBJECTS ON USER OPTIMIZATION

Network Cost (5 objects, 10 servers, 100 clients)				
Topology	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
Hierarchical	1.000	1.275	1.040	1.040
Random	1.000	1.568	1.029	1.028

TABLE XIII

EFFECT OF THE TOPOLOGY ON NETWORK OPTIMIZATION

User Dissatisfaction (5 objects, 10 servers, 100 clients)				
Topology	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
Hierarchical	1.000	1.256	0.972	1.006
Random	1.000	1.389	1.053	1.113

TABLE XIV

EFFECT OF THE TOPOLOGY ON USER OPTIMIZATION

Execution time in ms (5 objects, 10 servers, 100 clients)				
Topology	Net. Ideal	Random	**Prot.**	Prot. (no pri.)
Hierarchical	9750	50	794	750
Random	7000	47	895	840

TABLE XV

EFFECT OF THE TOPOLOGY ON OPTIMIZATION TIME

D. Network Topology

Table XIII shows that, in terms of network optimization, the proposed protocol works better in random networks. On the other hand, Table XIV shows that, in terms of user satisfaction, the proposed protocol works better (relative to network ideal mapping) in hierarchical networks. In either case, the protocol provides significant savings over random mapping. The network topology slightly affects the execution time of the algorithms. While for network ideal algorithm the execution time is higher in hierarchical networks, the protocol works sligthly faster in hierarchical networks (Table XV).

E. Summary

The results show that the use of the protocol reduces the cost of maintaining the system considerably when

compared with a random placement of objects and clients. The network ideal placement may be provide comparable results, however requires global knowledge about the networks and clients state and can not be implemented in a distributed manner.

V. Conclusion

In this paper we presented a protocol for server replication in interactive push-based content-delivery systems. The selection of a replica server should be such that the network and user satisfacation costs, associated with updates being propagated to the interested clients, are kept as low as possible. To achieve this, we introduced the concept of *server_score*. We identified the constraints affecting the system as well as the objective cost function. The relevant parameters included, client priorities for objects, rate at which clients update objects, and delays between clients and servers. We have experimentally evaluated the proposed protocols through simulations where the network conditions were systematically varied and the performance of the protocol was shown to be significantly better than a random placement of clients and objects and comparable to an ideal placement.

References

[1] A.Rowstron and P.Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, IFIP/ACM Int.Conf.on Dist.Sys. Platforms(Middleware), 2001.

[2] M. Castro, P. Druschel, A.-M. Kermarrec and A. Rowstron, *SCRIBE: A large-scale and decentralized application-level multicast infrastructure*, IEEE J. on Selected Areas in Comm. 2002.

[3] L. Liu, C. Pu, W. Tang. *CONQUER: An Architecture for distributed Push-enabled Data Management System*, Tech. Report, OGI/CSE, 1998.

[4] M.E.Crovella and R.L.Carter, *Dynamic Server Selection in the Internet*, 3^{rd} IEEE Workshop on the Arch. and Impl. of High Performance Comm. Subsystems, 1995.

[5] M. Mauve, J. Vogel, V. Hilt, W. Effelsberg, *Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications*, IEEE Trans. on Multimedia, 2002.

[6] Xiao Qin, *Delayed Consistency Model for Distributed Interactive Systems with Real-time Continuous Media*, Journal of Software, Vol.13, No.6, pp.1029-1039, 2002.

[7] S.Shirmohammadi and N.D.Georganas. *Collaborating in 3D Virtual Environments:A Synchronous Architecture*, WETICE, 2000.

[8] M.Hosseini, S.Pettifer and N.D. Georganas, *Visibility Based Interest Management in Collaborative Virtual Environments*, ACM Collaborative Virtual Environments Conference, 2002.

[9] C.B. Mayer, K.S. Candan, and V. Sangam. *Effects of User Request Patterns on a Multimedia Delivery System*, accepted for Multimedia Tools and Applications, 2004.

[10] V. Sangam, C. Mayer, and K.S. Candan. *Fairly Redistributing Failed Server Load in a Distributed System*, Workshop on Reliable and Secure Middleware, pp. 871-884, 2003.

[11] J.C. Oliveira, X. Shen and N.D. Georganas, *Collaborative Virtual Environment for Industrial Training and e-Commerce*, Chapter in Virtual Reality Technologies for Future Telecommunications Systems, Willey, West-Sussex, 2002.

[12] E.W.Zegura, K.Calvert and S.Bhattacharjee, *How to model an internetwork*, INFOCOM'96.

[13] Z. Fei, M. Ammar and E. Zegura, *Efficient server replication and client redirection for multicast services*, SPIE ITCOM Conf. on Scalability and Traffic Control in IP Networks, 2001.

[14] Jianliang Xu, Bo Li, Dik L. Lee,*Optimal Replica Placement on Transparent Replication Proxies for Read/Write Data*, Performance Computing and Communications Conference, 2002.

[15] Meng Guo, Mostafa H. Ammar, Ellen W. Zegura, *A Probe-Based Server Selection Protocol for Differentiated Service Networks*, IEEE Int. Conference on Communications, 2002.

[16] National Institute of Standands and Technology. http://www.nist.gov/dads/HTML/zipfian.html.

[17] Adobe Atmosphere. The 3D interactive stage set for the Web. http://www.adobe.com/products/atmosphere/main.html

[18] S. Jamin, C. Jin, A.R. Kure, D. Raz, Y. Shavitt, *Constrained Mirror Placement on the Internet*, INFOCOM 2001.

[19] S. Sivasubramanian, G. Pierre, M. van Steen, *A Case for Dynamic Selection of Replication and Caching Strategies*, International Web Caching Workshop, 2003.

[20] Jeffrey Pang, Justin Weisz, *Object Placement in Distributed Multiplayer Games* http://www-2.cs.cmu.edu/ jweisz/talks/F03/

[21] J. Kangasharju, J. Roberts, and K. W. Ross. *Object replication strategies in content distribution networks.* In Int. Workshop on Web Caching and Content Distribution, 2001.

[22] B. Li, M. J. Golin, G. F. Italiano, X. Deng, , and K. Sohraby. *On the optimal placement of web proxies in the internet.* IEEE INFOCOM 1999.

[23] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. *On the placement of web server replicas.* IEEE INFOCOM 2001.

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

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

[26] Digital Island Inc. http://www.digitalisland.com/

[27] Akamai http://www.akamai.com/

[28] Mirror Image Internet. http://www.mirror-image.com/

[29] S. Banerjee et al. *Construction of an Efficient Overlay Multicast Infrastructure for Real-Time Applications*, Infocom 2003.

[30] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. *Scalable Application Layer Multicast*, ACM SIGCOMM, Aug. 2002.

[31] R. Rajaie et al. Multimedia proxy caching mechanism for quality adaptive streaming applications on the Internet. INFOCOM 2000.

[32] Y. Wang et al. A network conscious approach to end-to-end delivery over wide-area networks using proxy servers. 1998.

[33] Z. Miao and A. Ortega. Proxy caching for efficient video services over the Internet. In *Proceedings of the PVW*, 1999.

[34] RealNetworks. *Information available at* `http://www.real.com/`.

[35] T. Nguyen and A. Zakhor. *Distributed Video Streaming Over Internet.* Proceedings of SPIE/ACM MMCN 2002, Jan 2002

[36] B. Ko and D. Rubenstein. *Distributed Server Replication in Large Scale Networks.* NOSSDAV 2004.

[37] Z. Ge, P. Ji, and P. Shenoy *A Demand Adaptive and Locality Aware (DALA) Streaming Media Server Cluster Architecture.* NOSSDAV 2002.