Journal of Network and
Computer Applications 26 (2003) 163–196

Journal of NETWORK and COMPUTER APPLICATIONS

www.elsevier.com/locate/yjnca

Experiences with an object-level scalable web framework

B. Prabhakaran[a,*], Yuguang Tu[a], Yin Wu[b]

[a]*Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083, USA*
[b]*Department of Computer Science, National University of Singapore, Singapore, Singapore 117543*

Received 11 September 2002; received in revised form 5 February 2003; accepted 13 February 2003

Abstract

We present an object-level scalable web framework and discuss our implementation as well as simulation experiences with them. This object-level scalable web framework automatically monitors access patterns, replicates, and maintains large multimedia objects among a set of geographically distributed web servers without the need for full URL replication. This framework employs a *traceable RTSP/HTTP redirection* approach to avoid cyclic redirections among the (geographically distributed) web servers.

We implemented this web framework on a wide area network with a set of three web servers: one at the Technical University of Darmstadt (Germany), second one at the Virginia Polytechnic University (USA), and the third at the National University of Singapore. We also carried out a series of simulation experiments to analyze the scalability of this object-level scalable web framework. In this paper, we share these implementation and simulation experimental results.

Keywords: Redirection; Web framework; Scalability

1. Introduction

A dramatic increase in the number of users, coupled with multimedia based information services, real-time audio/video transmission and the booming of electronic commerce have raised great demand for a scalable, high-performance Internet server system. A distributed architecture consisting of several independent servers jointly providing services is an accepted solution to provide high scalability for popular websites (Andersen

* Corresponding author.
E-mail address: praba@utdallas.edu (B. Prabhakaran).

1084-8045/03/$ - see front matter
doi:10.1016/S1084-8045(03)00016-X

et al., 1996; Cardellini et al., 1999; Colajanni et al., 1998; Fox et al., 1997; Schulzrinne H; Yang and Luo, 1998). Usually, the collaboration among servers is transparent to outside world and hence the end users only access a single URL to get service from a distributed Web system. A considerable number of dispatching and routing strategies in such system have been proposed to collaboratively distribute user requests.

In this work, we consider web servers handling large multimedia objects such as movies or multi-media lectures. We propose an object-level scalable framework that can monitor access patterns, replicate, and maintain large multimedia data at the object-level (as opposed to URL-level) among servers. These servers can be distributed in a Wide Area Network. The framework can support both Real-Time Streaming Protocol (RTSP) and HTTP. Routings of incoming requests are handled in two stages: first using a DNS and then using HTTP/RTSP redirection to handle requests based on servers' workloads and geographical area from which requests are coming in. We used RTSP/HTTP redirection since other approaches such as packet rewriting or TCP splicing incur too much over head per data packet in a wide area network of servers. Other features of the proposed scalable web framework are

- The identification of objects to be replicated is dynamic and at the object-level according to access patterns. This set of *hot* objects gets dynamically updated based on access patterns. The framework monitors access patterns at the object level (movies, lecture presentations, product catalogues, and other large multimedia objects). It identifies hot objects that cause increase in server workload and replicates them in some selected servers based on their workload as well as geographical nature of access patterns. For instance, a movie that is normally delivered by a server in the US may be replicated in a server in Europe based on access patterns. Also, less popular movies may be removed from replicated servers to make way for the more popular ones. An identified hot object may have a set of *dependent objects* that might be accessed along with the hot object. For instance, a multimedia lecture may have associated objects such as homeworks, projects, or a set of reference materials. A movie may have a set of associated objects such as trailers, production information, or statistics. We use a HTML parser to identify these dependent objects and replicate the dependent objects along with the hot object as well.
- The routing of requests is at the object-level in terms of the contents of requests and current server workloads. Especially, the times of redirections for a RTSP/HTTP request in a transaction are decided by the sizes of objects because there is an overhead for redirection. For instance, a small object cannot be redirected or only can be redirected few times.
- We have designed a *traceable request redirection* approach to prevent cyclic redirections. Since we use RTSP/HTTP redirection among servers in a Wide Area Network, the redirections can become cyclic in nature. Each RTSP/HTTP request in this scalable framework is modified to carry a record of its *redirection history*. The traceable request redirection approach examines the redirection history before carrying out RTSP/HTTP redirections.

We tested the proposed framework with a distributed implementation as well as through simulation. The distributed implementation was over a set of three web servers:

one at the Technical University of Darmstadt (Germany), second one at the Virginia Polytechnic University (USA), and the third at the National University of Singapore. We carried out performance measurements using a client in the National University of Singapore that can send multiple requests to a web server by opening multiple HTTP/RTSP connections. We also did some measurements with a simple web client in China. These measurements indicate that the traceable HTTP/RTSP redirection approach provides a perceivable improvement in the response time. To test the scalability of the proposed framework, we also carried out a series of simulation runs.

This paper is organized as follows: we consider the related work in this research area, in the following section. In Section 3, we describe the design of the proposed object-level scalable web framework. We then discuss the implementation of this framework, in Section 4. We present the performance of the distributed implementation of the scalable web framework in Section 5. Simulation results of the object-level scalable web framework are outlined in Section 6.

2. Related work

A number of commercial products and researches that offer transparent request distribution among multiple mirrored web servers use one or more of the following approaches:

- *DNS-based selection*. A modified Domain Name System (DNS) server selectively maps the URL-name to IP-address of one of the back-end servers (Andersen et al., 1996; Kwan et al., 1995). However, IP-address caching mechanism at the intermediate name servers limits the authoritative DNS server's control over the incoming requests.
- *Packet rewriting*. Once the server's IP-address is resolved, the client needs to set up a TCP connection with the server. In some products (e.g. IBM Network Dispatcher (Hunt et al., 1998)) and research (Dias et al., 1996), an IP dispatcher is used to rewrite each incoming packet's destination IP with the IP of one of the backend servers. In this model, the IP dispatcher may be the bottleneck and cannot scale easily to a WAN distributed web system.
- *HTTP redirection*. After the TCP connection is set up, the client sends out a HTTP request. At this stage, the HTTP redirections among servers can be used to reassign the user request (Andersen et al., 1996; Cardellini et al., 1999) and to help alleviate the side effect of caching URL-name to IP-address mapping. This redirection mechanism is achieved by using a special response code defined in the HTTP protocol (Fielding et al., 1999).
- *Application Layer Anycasting*. The anycasting (Fei et al., 1998) communication paradigm is designed to support server replication by allowing applications to easily select and communicate with the 'best' server, according to some performance or policy criteria, in a group of content equivalent servers. It is appealing because it can avoid burdening links with repeated requests to gather server distance information. However, a downside is that anycasting assumes all servers provide equal services and host same contents.

Full replication of all the content among all the backend servers is most widely used in many of the above scalable (Andersen et al., 1996; Cardellini et al., 1999; Colajanni et al., 1998; Fei et al., 1998; Hunt et al., 1998). This is expensive in terms of space utilization and transfer cost, and every server must have the same capability of processing all kinds of requests coming to any portion of the content the website provides. Especially, multimedia data consume enormous storage and bandwidth, hence this would expect the system to have very large disk farm for each hosting server. However, a recent study about web characterization shows that there is a high concentration of references for the web contents (e.g. 10% of the files accessed on the server typically account for 90% of the server requests and 90% of the bytes transferred (Arlitt and Williamson, 1997)).

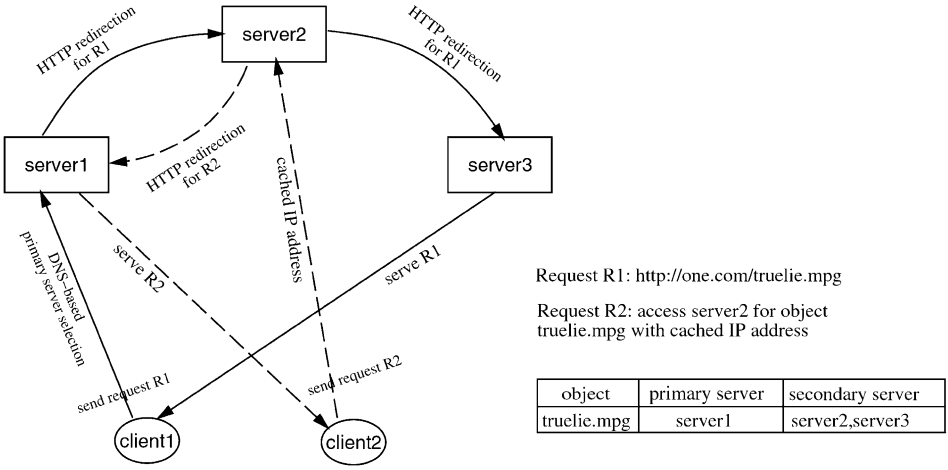
2.1. Dynamic replication

The idea of dynamic replication has been extensively studied in the database area, but Internet gives it a completely different focus. Much of existing work on dynamic replication has concentrated on maintaining replica consistency. This work focuses on algorithms for adjusting quorums when replica sets change (Jajodia and Mutchler, 1990; Rabinovich and Lazowska, 1992) and on regenerating new object replicas in place of failed ones (Long et al., 1989). In contrast, our replication and migration mechanism is mainly for improving performance. Consequently, the focus shifts to issues such as where to place the replicas, when to perform the replication or migration, how to decide the replica set and how to route requests among these replicas. Wolfon et al. (1997) addresses the performance of distributed database systems. It proposed an ADR protocol that dynamically replicates objects to minimize communication costs due to reads and writes. However, recent trace studies (Manly and Seltzer, 1997) show that most of Internet objects are rarely written, so this is not a suitable cost metric for the Internet. In addition, the protocol imposes logical tree structures on hosting servers and requires requests to travel along the edges of the tree. This logical topology is not really matching with the real network topology, so it would result in high delay of request propagation. Moreover, in ADR, objects are replicated only between neighbor servers, which would result in high overheads for creating distant replicas. Our system performs replication directly between two distant servers. In the web services provided by Akamai (Fast Internet content delivery with free-flow), the servers use DNS based redirections to allow clients access the *nearest* possible cache.

While dynamic replication is very similar to push caching (Gwertzman and Seltzer, 1994; Tewari et al., 1998), caching has some limitations, which necessitates the need for replication. Sometimes service providers want to disallow caching for the reason of copyright protection, pricing purpose, usage metrics, and etc, even when these objects are cacheable. Also caching only provides a weak consistency check by HTTP protocol while replication can utilize arbitrarily strong consistency guarantees. Caching, therefore, is complimentary to our replication solution. Our system processes the requests that are missed in the caches.

3. System design

The object-level scalable architecture is shown in Fig. 1. This system consists of a set of distributed servers that manage different web contents. The system can monitor access patterns, replicate and maintain large multimedia data among these servers. The framework can support both RTSP and HTTP conceptually. One URL may be served by a set of *primary* and *secondary servers*. Primary servers are the origin web servers and have entries in the authoritative DNS servers for the corresponding URLs. One of the IP addresses of the primary servers will be mapped to the requested URL-address by DNS. Secondary servers are those that only host some hot objects of this URL for their respective regions. It should be noted here that one server could function as a primary server for a set of URLs and as a secondary server for another set of URLs at the same time. For example, server1 is the primary web server for the object http://one.com/truelie.mpg in Fig. 1. When this object becomes very popular in the region where server2 is located, this hot object truelie.mpg will be automatically replicated to server2. When a new request comes in for truelie.mpg, server1 might redirect it to server2. Now, if the load on server2 is such that the redirected request for the movie cannot be admitted, server2 may redirect the request to server3 (assuming that server3 also serves the movie). There are two main issues to be addressed for the object-level scalable architecture. First, whenever an object is replicated or removed, the new object assignment should be updated on every server, so that new requests can be redirected appropriately. An *object assignment table* is maintained in each server for this purpose. This object assignment table contains object-to-server(s) mapping information. Since in the above example, the object truelie.mpg is replicated from server1 to server2 and server3, the corresponding mapping entry in the object assignment table on server1 and server2 will be updated as shown in Fig. 2 In addition, a multi-faceted algorithm is needed to make decision on what and where to move when the object replication is needed. We discuss this issue in detail in Section 3.1.



object	primary server	secondary server
truelie.mpg	server1	server2,server3

Fig. 1. Object-level scalable architecture.

Object Type	Object Name	Hosting Servers
Local Object	/web_home/truelie.mpg	server1
		server2
		server3
Replicated Object	–	–

object assignment table on Server1

Object Type	Object Name	Hosting Servers
Local Object	–	–
Replicated Object	/mirror/server1/truelie.mpg	server1
		server2
		server3

object assignment table on Server2

Fig. 2. Object assignment table.

Secondly, the second-level RTSP/HTTP redirection mechanism we have used is part of the RTSP/HTTP protocol and is currently supported by all popular browsers and server software. Such redirections are transparent to users. The browser can automatically recognize the redirection message, derive the new location from the RTSP/HTTP header and connect to the new server to fetch the object. However, cyclic redirections among the servers might happen because the RTSP/HTTP protocol is stateless. Therefore, we have designed an efficient method to keep track of the redirection history to avoid a ping-pong effect. This traceable request redirection method is described in Section 3.3.

For ease of reference, the Table 1 summarizes all the notation used in this paper. Some of the notation given in Table 1 will be defined later in the paper, as it is needed.

3.1. Dynamic object replication

The update of object assignment table can be triggered by three events: hot object replication, cold object removal, and object update. In each of these cases, the new mapping from an object to its hosting servers in the object assignment table must be updated on every hosting server. We use primary server-centered approach for handling object replications where it is the responsibility of the primary server to update the object assignment table in corresponding secondary servers.

Table 1
Notation summary

S_x	A replica of object s on server x
$load_x(t)$	The load on server x at time t
$load(s_x)$	The load on server x due to object s
LW	Low watermark for server workload
MW	Maximum capacity of server workload
h	Replication threshold limit, the trigger for replication process
ϕ	Hard limit of replication bandwidth

3.1.1. Hot object replication

Replicating highly accessed large objects to a server in popular regions can greatly improve the response time. Also, the availability is improved because a single server failure no longer eliminates all accesses to data. However, dynamic replication could result in significant system overheads, especially for large multimedia objects, e.g., in form of additional disk bandwidth, network bandwidth and memory buffers. Hence, we must address the following issues:

(1) *When to move.* We adopt a threshold-based policy for triggering dynamic replication. The *threshold limit parameter h* is introduced for this purpose. It is defined as percentage of server capacity. The impact of parameter h will be illustrated in Section 6. If $Load_x(t)$ increases above hMW, then the host will try to send replication request to other servers. As discussed, the threshold limit h needs to be chosen carefully to control excessive replications. It is a tradeoff between dedicating more system resources for replication and dedicating these resources for normal processing of arriving requests. In the simulation study, we will show that when h decreases from 0.9 to 0.1, more replicas are created. However, at low threshold limit the performance of the system starts degrading with decreasing threshold limit.

In our system, each server maintains an estimate of its load. The load estimate is updated and exchanged among servers periodically. This update interval cannot be too short since it will cause high computation and communication overheads. In our simulation study, we demonstrate the effect of changing the update interval to the system performance. We will assume that load metric represents a single computational component, e.g. the number of requests handled by the host. In general, the load metric may be represented by a vector reflecting multiple components, notably computational load and storage utilization.

(2) *What to move.* Once a server reaches the threshold limit, it will go through all its objects, starting with those that impose higher workload to the server and attempts to replicate them. These objects are called as *hot objects*. The load due to an object $load(s_x)$ is calculated as the bandwidth consumed by the object, which is the result of total effective transmission sizes for this object over a period of time. This parameter is also used as the additional workload by the Replica Placement algorithm. The server continues in this manner until its load drops below hMW. The access log for a web server can be used to understand geographical access patterns for large multimedia objects. When the access patterns indicate that an object has become very hot and imposed a lot of workload to the server, then this object will be replicated to the hot region.

In our simulation model, an object access statistics table is maintained on each server to keep track of the object popularity; and when the server reaches the threshold limit it will look up this table for the hot objects. In the real world scenario, there may be *dependant objects* such as embedded objects or link objects associated with a hot multimedia file. We use a HTML parser (described in the chapter of Implementation Experience) to identify these dependant objects so that they can be replicated along with the hot object. These dependant objects are added into the object assignment table as well.

(3) *How to choose source server and destination server.* The source server is chosen from the least loaded servers which host the replica since the replication would result in an additional load on the source server. There are several performance factors that affect our

decision on the destination server selection:

(a) server disk I/O, which limits how many requests the server can admit
(b) disk capacity, which limits the size and the number of the objects the server can host simultaneously
(c) geographical access patterns, which reflect the popularity of the object by regions
(d) the additional workload (for example, disk I/O consumed by the object and the size of the object) caused by moving the hot object to the destination server. This prevents the situation that the replication would cause the destination server to be overloaded
(e) information, if available, on network characteristics such as bandwidth.

The first two factors can be derived by using a local resource monitor on each server to observe its own status and multicast this information to the other servers. Factors (c) and (d) can be tracked by the access log on the web server. Factor (e) can be obtained through router database. Thus, our algorithm is multi-faceted in the sense that the proper decision on choosing destination server needs to aggregate the impact of the above factors on the overall system response time.

Replica Placement Algorithm

Our algorithm has two major goals:

- Increase client server proximity: Replicas should be placed in such a manner that they become closer to the clients, thereby reducing latency and bandwidth consumption.
- Distribute load among servers: The replicas should be placed keeping in mind the load of the hosts in the systems. Migrating a large number of replicas to a particular host might increase its load beyond acceptable limits.

To combine these two goals, in our algorithm the overloaded host will send a replication request to the *closest server whose load is below a threshold*. This simple and intuitive approach takes care of proximity and at the same time distributes load. However, we observed this algorithm suffers from a problem of *competition for service capacity* by multiple replications. This can be observed from the fact that there are several object replication processes going on at the same time and all these processes are trying to send requests to the same target server. In order to prevent simultaneous replications of different objects to the same target server, which may lead to a temporary overloading on this server, we use a probabilistic algorithm (Table 2). If the load of the server is below LW, we send the request to it. However, if it is between LW and hMW, we send the request to the host with a probability that is a function of the current server load. Our simulations show that this modified algorithm successfully reduces the likelihood of competitions for service capacity.

To replicate object s_x to a candidate server y, x sends a request to y, which includes the ID of the object and the load on host x generated due to object s. The candidate y accepts a replication request if its load is below low watermark and its upper-bound load estimate after the proposed replication would be below hMW. This prevents a situation where the replication might bring the recipient's load to be above the replication threshold limit. In addition, the network bandwidth in server y local area should be enough for performing

Table 2
Algorithm for replica placement

```
ChooseDestinationServer( )
/* Executed by server x */
for each candidate server y in the server list of non-hosting object s in order of increasing distance
 if load_y(t) < hMW then
  if load_y(t) < LW
    send replication request CreateObj(s, load(s_x)) to server y
    if y responded with 'OK'
     break from loop
    endif
  else
    send replication request CreateObj(s, load(s_x)) to server y with probability
    proportional to (hMW − load_y(t)) / (hMW − LW)
    if y responded with 'OK'
     break from loop
    Endif
  Endif
 Endif
Endfor

CreateObj(s, load(s_x))
/* Executed by candidate server y */
 if load_y(t) > LW
  send 'Refuse' to request host x and terminate
 endif
 if load_y(t) + load(s_x) > hMW
  send 'Refuse' to request host x and terminate
 endif
 if nw_bw(y) is not enough for replication
  send 'Refuse' to request host x and terminate
 endif
 send 'OK' to server x
 copy object s from host x
 load_y(t) = load_y(t) + load(s_x)
 nw_bw(y) = nw_bw(y) − ReplicationBandwidth( )
 if the new copy of s is successfully created
  notify the primary server of this object to update the object assignment table
 endif

ReplicationBandwidth( )
 return min(nw_bw(x), nw_bw(y), ϕ)
```

the replication. If these conditions are satisfied, the server y creates a copy of s and then notifies the primary server for this object to update object assignment table.

(4) *How much disk bandwidth for replication:* this issue is directly related to how to reduce the replication time. The tradeoff here is between dedicating more system resources for replication in order to put a new replica into work (and thus help give clients better response time) versus dedicating these resources for normal incoming clients' requests. Therefore, a replication bandwidth limit ϕ is used for this purpose. The replication

bandwidth would be the minimum value among available local network bandwidth for source server and destination server and the decided replication bandwidth limit. The resources will be utilized for the time

$$\frac{\text{size of object}}{\text{replication band width}}.$$

(5) *How to avoid name collision.* Name collision occurs when two servers A and B have replicated objects with the same name from different websites into the same directory on server C. This might result in files being overwritten. To avoid this, replicated objects can be put into a directory which indicates where the object is originally from. For example, the object truelie.mpg from server A and server B can be generated as /mirror/server-A/truelie.mpg and /mirror/serverB/truelie.mpg, respectively.

3.1.2. Cold replicated object removal

When the popularity or hits to replicated objects come down, we may remove it to make space for other hot objects. A situation such as a recently removed object being replicated back again is undesirable, so we need to observe the object access behavior for sufficiently long time and make better prediction. In our design, we set a variable RemoveDelayCount for each replicated object to keep the track of the popularity (Table 3). This variable is updated when the hot object table is checked periodically.

The variable RemoveDelayCount has a maximum value in case the value becomes too large when burst accesses happen. At the same time, in order to prevent a vicious cycle of replica creations and deletions, a minimum value is set at the first time when the object is added into the object assignment table.

Before a replicated object is removed, permission must be obtained from its primary server. Otherwise, the primary server may become abruptly overloaded if secondary server(s) stops serving the replicated object and transfers the workload back to it. Another reason to inform the primary server is the need for updating the object assignment table

Table 3
Algorithm for cold object removal

```
for each (replicated object)
 if (object in HotObjectTable) then
  increase RemoveDelayCount;
 else
 decrease RemoveDelayCount;
  if (RemoveDelayCount = = 0)
   if (primary server for this object permit deleting this object)
    delete this object and update the object assignment table
   else
    keep this object for another period of time
   endif
  endif
 endif
endfor
```

reflecting the removed entry and for multicasting the new object assignment table to other servers.

3.1.3. Object update

Consistency of an object has to be maintained on all the hosting servers when users send 'put' or 'delete' request to update objects. Traditional method of updating whole websites on all servers is very inefficient, especially if only a small number of objects are updated. In our framework, update procedure is done using the object assignment table without the need for manual intervention. When an update request comes in, the primary server of the object to be updated will first either update or delete the requested object from its file system. Then, it checks the object assignment table and informs all the secondary servers for this object to take the corresponding action. For the 'update' action, there is no need to update the object assignment table. For the 'delete' action, the primary server needs to delete this object entry from the assignment table and multicast this information to the other servers.

Updating object assignment table. After replication or deletion of an object, only the primary server of the replicated/deleted object multicasts the local objects' assignments. After the primary server sends out the multicast message, it starts a timer and waits for the acknowledgement from the secondary server(s). If the primary server does not receive an acknowledgement from a secondary server until the time out, then it assumes that secondary server is unreachable. Then, the primary server removes that server from the object-to-server mapping and re-multicasts the object assignment table. After a server recovers from its temporary crash, it checks its replicated object with the corresponding primary server and checks its local object with the secondary servers.

3.2. Content-based request routing strategies

A good number of routing strategies are proposed by researchers. Some of them aim at improving the workload balancing among servers (Andersen et al., 1996; Cardellini et al., 1999; Kwan et al., 1995). Some target at locating the request to the nearest server (Guyton and Schwarts, 1995; Sayal et al., 1998). In most of these approaches, full replication among all servers is needed and thus every server must have the same capabilities of handling all kinds of requests. In our framework, we take the nature of the requested object into account for routing. The object-to-server mapping is maintained on servers, thus the request needs special service such as video presentation support can be routed to specific servers. Hence, the system does not require every server to have the same capability. Moreover, the redirected workload imposed on a destination server is not only decided by the number of hits (requests), but also by the size and nature (e.g. video) of objects.

Content-based routing strategies in our object-level scalable web framework are using RTSP/HTTP redirections among servers. This is provided by RTSP/HTTP protocol and completely transparent to end users. Depending on the type of information used, we can partition redirections into two main classes:

- *Client-server proximity based redirection*. In this policy, the geographical access patterns of requested objects will be tracked down. Then, the request is always

redirected to the nearest server without considering the server workload. In our simulation study, the geographical proximity is determined by the number of hops between the client's domain (local gateway) to the server. This value is relatively static (Paxon, 1997) and can be obtained directly from the routing table without incurring any additional network load. The simulation result shows this routing policy is effective when request rate is low and it exhibits poorly when request rate increases. This is because the network topology often does not correspond to geographic proximity and this policy is not responsive to the server workload.

- *Server workload based redirection*. In this policy, the request is always redirected to the server with the minimum workload. Therefore, we always redirect the request to the server with the maximum available load or the server with the shortest waiting queue if all servers are busy. In the simulation, we find that one problem with this algorithm is that when a new replica is created, it will be chosen for all requests until its request count catches up with the rest of the replicas. This may cause a temporary overloading of new replicas. To avoid this, the primary server for this object resets all request counts to 0 whenever it is notified of any changes to the replica set for the object.

When performing the redirection, the times of redirections for an object are determined by the size of objects. Small-size of objects cannot be redirected or only can be redirected few times. In addition, the identification of hot objects is also taken the size of objects into account. Large-size of objects are given higher priority.

3.3. Traceable request redirection

Decisions of the algorithm used for selecting a server for an incoming request may not be very accurate due to varying system (network and server) loads. Therefore, multiple redirections may be needed, though they have a couple of problems. First, they introduce more overhead than single redirection and the client's response time can suffer. However, when we consider requests for large objects such as video or multimedia lectures, this redirection overhead can be very small compared to the request service time. Second, a request may be redirected among servers in a cyclic fashion. This problem is particularly serious when most of the servers serving the same object are overloaded.

Hence, we have added redirection control to coordinate our servers' behaviors so that one request will not be redirected to the same server twice. This is done by tracing the request redirections.

3.3.1. Redirection history

Our idea is to eliminate cyclic redirections while allowing multiple redirections. Therefore, we need to uniquely identify a redirected request so as to keep track of its redirection history. Three parameters are needed for redirection history:

1. a list of servers to which the request has ever been redirected;
2. the time of the last redirection;
3. a request ID.

The time of redirection is helpful to identify the requests that have been cached. In that case, the request has been already served before. The system then would refresh the redirection history and serve the request as if it has never been redirected.

3.3.2. How it works?

The request redirection is done by taking the advantage of code 302 (Temporary Redirect) in HTTP1.1. A server (that cannot handle the incoming request due to high load) inserts the redirection history information into the value field of the Location in the HTTP response header, which follows the command 302 to indicate the new location of the requested file. Upon receiving response 302, the client(browser) will terminate the current connection and connect to the new location of the requested file.

The client carries the redirection history to the destination server together with its redirected request. Take the server response in Table 4 (last tuple) as an example. The destination server (server2) will interpret the underlined part in the following way:

1. *RH*: (redirection history) Header of a redirection history
2. *137.132.101.27*. IP-address of server1
3. $>>$: Delimiter
4. *Jan-28-14-02-04*: The time of last re-direction. This time is converted to Standard Greenwich Mean Time to take care of the time difference among regions
5. *S0135*: The request ID
6. *mirror/truelie.mpg*: The actual path of the file.

The overheads introduced by multiple redirections are usually negligible since our system is mainly targeted at the web sites with large multimedia files. Requests for small objects are not allowed to be redirected for many times. Also, requests that have been redirected for more times are assigned higher priorities.

Table 4
Comparison of server responses with redirection history and without redirection history

Server response	Request to server1	Response from server1	Request to server2
Without redirection history	Get /truelie.mpg HTTP/1.1	HTTP/1/1 302 temporary redirect location: http://server2/mirror/truelie.mpg	Get /mirror/truelie.mpg HTTP/1.1
With redirection history	Get /truelie.mpg HTTP/1.1	HTTP/1.1 302 temporary redirect location: http://server2/RH 137.132.101.27 $>>$Jan-28-14-02-04 $>>$35/mirror/truelie.mpg	Get/RH137.132.101.27 $>>$Jan-28-14-02-04 $>>$S0135 $>>$/mirror/truelie.mpg HTTP/1.1

4. Implementation

The proposed system is implemented as an object-oriented framework using Java JDK1.2. The development environment consists of Sun Solaris 7 and Windows NT 4.0. These hosts are identified by their network (IP) addresses. There are two main components to our scalable server: a HTTP/RTSP server and a replication service. The HTTP/RTSP server is responsible for serving or redirecting the clients' requests according to geographical access patterns and server workloads, and tracking geographical access information for the hosted files. The replication service keeps track of the status of servers, the amount of available free space on each server and each server's workload. The replication service works with the HTTP/RTSP server to decide where to place replicas. It also runs an ftp server to replicate objects (i.e. transfer files). A server in the proposed scalable architecture has the following modules (Fig. 3).

4.1. Request handler

Consider an example where a client selects a movie. A RTSP request for that movie file comes to a server. Then, request handler on that server needs to decide whether to serve or redirect it with redirection history to another server based on geographical access patterns and server workloads. In our implementation, each RTSP request is an instance of class RTSPRequest (Table 5). Before request handler redirects a request, it first checks if the request is *redirectable*. A request is not redirectable under the following conditions:

- The new destination server for redirection is in the list of redirection history (i.e. cyclic redirection would happen). In this case, the request handler has to choose another destination server or the current server serves the request if there is no other better choice.
- The maximum redirection limit is reached. From the redirection trail, we can know how many times the request has been redirected. (Actually, in our program we set a field for counting the number of redirection times in class RTSPRequest for easier reference.) Since multiple redirections introduce more overhead than single redirection, the client's response time can suffer if a single request has been redirected many times. The maximum redirection limit in our program is determined in terms of the size of objects. If the number of redirections for a request reaches this limit, it may indicate that most of the servers serving the same object are overloaded in the system, so further redirection may just cause more overhead without improving the response. Therefore, the current server serves this request.

An example of redirection with redirection history is shown in Table 4). The server2 interprets that the request of ID S0135 is redirected by server 137.132.101.27 at 14:02:04 p.m. Jan 28th. It then checks the last redirection time and the ID for *turn-back*. A turn-back means that the same request (request with the same ID and time) has already been served before. This is possible in case that the client caches or bookmarks the temporary URL (i.e. the URL containing the redirection history. e.g. http://server2/RH137.132.101.27 > >Jan-28-14-02-04 > >35/mirror/truelie.mpg) and turns back

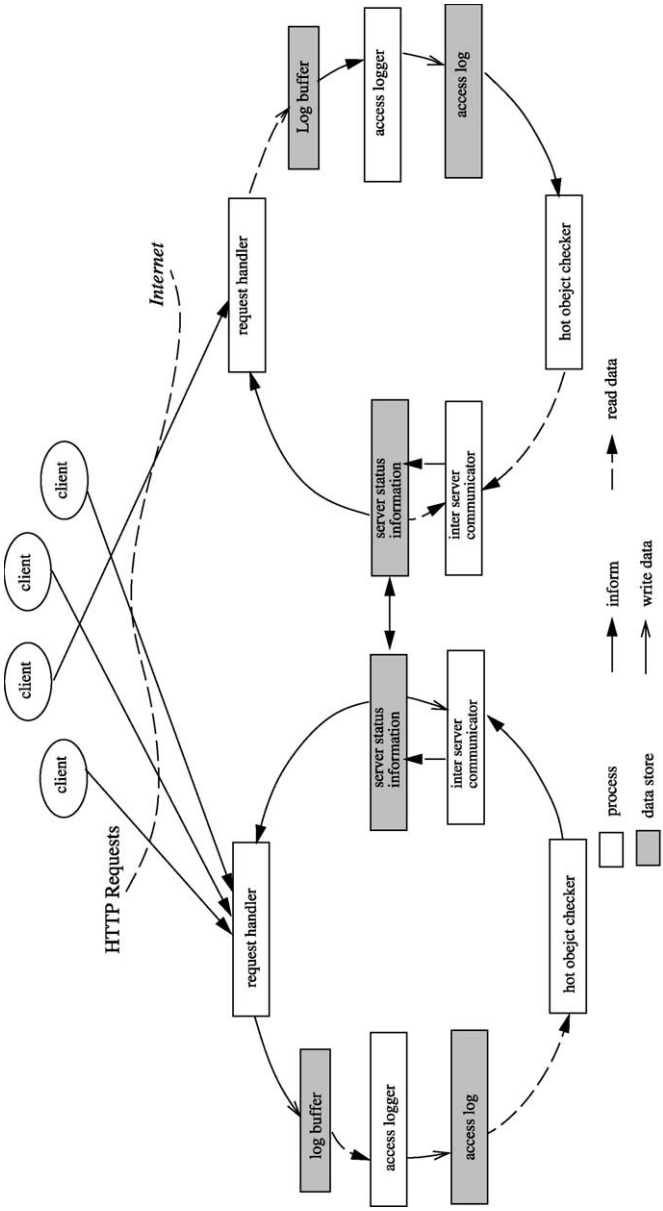


Fig. 3. Top level design showing interaction among modules.

Table 5
Structure of class RTSPRequest

```
class RTSPRequest
{
 static final String REDIRECTION\_HEADER = 'RH';
 long lastRedirectTime;
 int redirectTimes; String[ ] redirectionTrail;
 String objectName;
 String primaryServer;
 Boolean localObject;
 String protocol;
 Vector headerNames, headerValues;
 Boolean mime;
 public RTSPRequest( ); initialize function
 ...
}
```

some time later. A turn-back request is actually a new request with old redirection history and ID. In case of a turn-back, we refresh the redirection history and serve the request as if it has never been redirected. If the request is not a turn-back, then it is regarded as a normal redirected request. Then server2 will either serve the request if it is capable of doing so or further redirect the request if it is currently busy. No cyclic redirections will happen since we are fully aware of the servers that have been redirected to. In the above example, the response of server2 may be:

- *Case1 Turn-back & Server available*: HTTP/1.1 200 OK… (start serving)
- *Case2 Turn-back Server busy*: HTTP/1.1 302 Temporary Redirect Location: http://server3/RH137.132.89.224 > >Jan-28-14-02-06 > >S0201/mirror/truelie.mpg
- *Case3 Normal Server available*: HTTP/1.1 200 OK… (start serving)
- *Case4 Normal Serve busy*: HTTP/1.1 302 Temporary Redirect Location: http://server3/RH137.132.89.224 < 137.132.101.27 > >Jan-28-14-02-06 > >S0135/mirror/truelie.mpg

4.2. Access logger

When request handler serves a request, it will write the information related to this request in a temporary buffer in memory at the same time. As shown in Fig. 4, the access logger periodically checks this buffer, calculates and updates the access patterns in the statistics table, and at the same time, puts the hot object into a hot object table, and then purges the buffer content into a log file.

4.3. Hot object checker

Hot object checker is responsible for checking the hot object table periodically and coordinating with inter-server communicator to replicate or remove objects. The hot object replication scenario is shown in Fig. 5. According to our primary server-centered

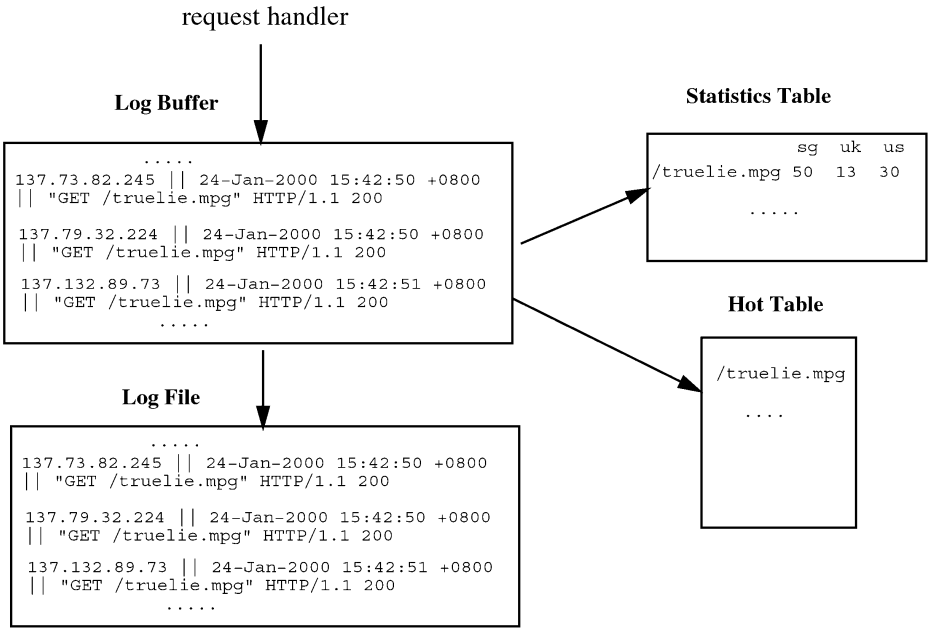


Fig. 4. Access log update.

design discussed in Section 3, if the server is the secondary server for a hot object, it will report the hot object to its primary server. If the server is the primary server for the object, then no reporting message is needed. Then, the primary server uses a HTML parser to identify the dependant objects that should replicate together with this hot object. Based on factors such as the additional workload caused by this object, the current workload of all the servers and the access pattern for this object, the primary server chooses a candidate server and sends a replication request message to it. If this candidate server accepts the request, then the hot object associated with its dependant objects will be replicated to it. This candidate server becomes the secondary server for the replicated object. The primary server updates this new object-to-server mapping information in the object assignment table. If this candidate server refuses to handle the hot object, then the primary server tries the second candidate and so on. This procedure will stop when one of the candidates accepts the replication request or all the appropriate candidates have been contacted. The object assignment table is implemented as two hashtables indexed by object names. One table is for local objects, the other is for mirrored objects. When the primary server multicasts object assignment table, only the local table is multicasted.

4.4. Object updates

The scalable web servers framework also provides facilities for updating the hosted objects. After authentication, a server to which the client is attached automatically performs file update among participating servers that host the object. We add a lock to the hot object being updated so that no two critical actions (such as replication, removal and

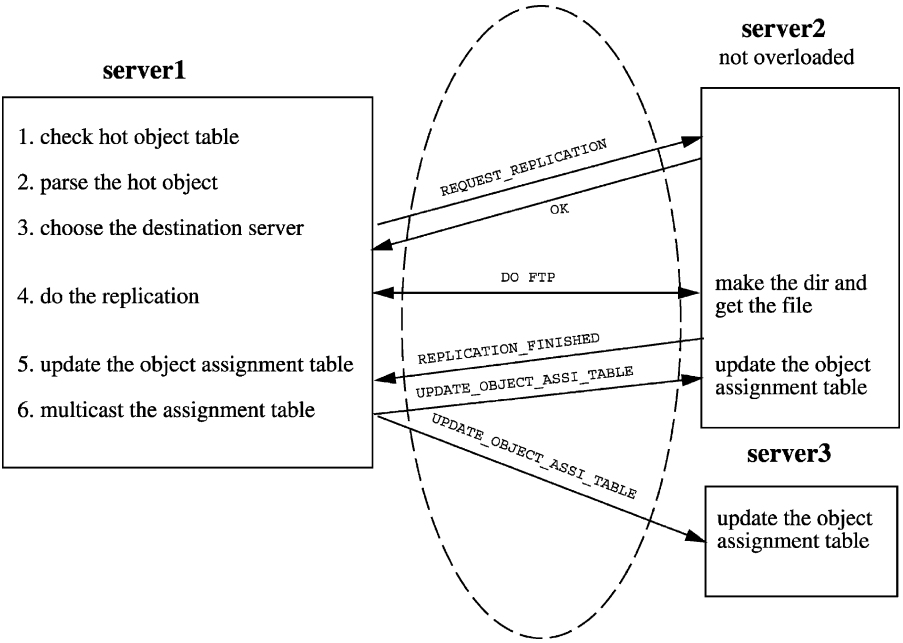


Fig. 5. Hot object replication scenario.

updating) can be done on the same target file simultaneously. This is to eliminate the scenario in which a file is removed in the middle of file replication. (Pseudo-code for file removal is shown in Tables 6 and 7).

4.5. Integrating with existing products

Our system has been tested with some of popular web browsers. We observed that different browsers react differently to the HTTP command 302 (Temporary Redirection). Take Internet Explorer 5 (IE5) and Netscape 4.7 as examples. IE5 hides the redirection process from users, whereas Netscape loyally reflects the redirection process by displaying the new URL (the URL containing the redirection history) in the address bar each time its request is redirected. As a result, IE users will not be aware of the redirection at all, but Netscape users may sometime bookmark the temporary URL (e.g. http://server2/RH137.132.101.27 > >Jan-28-14-02-04 > >35/mirror/truelie.mpg). Our system is able to handle this difference. The bookmarking of temporary URLs is treated as a turnback (Section 4.1) in the traceable request redirections. In the current implementation, we developed a web server by ourselves such that it does not depend on any specific products. Since our system is modular, the additional functional modules for object-level scalability are independent of the basic functional modules of normal web servers. The current popular web servers such as Microsoft IIS and Apache web servers all have access logs, and provide an interface library for the developers to enhance the web server function. So, we can easily use the existing web servers for our framework by adding a patch.

Table 6
Removal on primary server

```
if (DELETE request comes from client and file is a local file)
 if (file is NOT in object assignment table)
  delete local file
 else
  lock(object);
  remove file from object assignment table;
  remove local file;
  request secondary server to remove this file;
  unlock(object);
else reply error
```

5. Performance measurement

In order to test the performance of our framework, we installed three servers: one in the Technical University of Darmstadt, Germany, second at the Virginia Polytechnic University, USA, and the third at the National University of Singapore. We tested this distributed implementation with a client at the National University of Singapore that sends multiple requests over multiple HTTP/RTSP connections. We also used a client from China. The detailed configuration of servers are shown in Table 8. The client in China connects to Internet through 36.6K USRobotics Modem. The client in Singapore connects to Internet through NUSNET-III LAN.

We measure redirection time, which is the time from one server (e.g. yurong) receiving the client request to another server (e.g. video) receiving the redirection request from the previous server (yurong). This time includes the server processing time and RTSP/HTTP connection time to the second server. In addition, we measure the download time with redirection and without redirection. The download time is the time from the client sending out requests to fully downloading that request page (with embedded image). This term includes the HTTP connection time, redirection time, server processing time, queuing time and downloading time.

5.1. Experiments from Singapore

In Singapore, we implemented a dummy client that can send multiple requests to a web server by opening multiple HTTP/RTSP connections. This client observes RTSP/HTTP

Table 7
Removal on secondary server

```
if (DELETE request comes from primary server and file is a mirror file)
 lock(object)
 remove file from object assignment table
 remove file from disk
else reply error
```

Table 8
Hardware configuration of experimental servers

NAME	yurong	video	test04
Location	NUS, Singapore	Virginia, USA	Darmstadt, Germany
Model	Sun Ultra 5	Sun Sparc 10	Sun Sparc 10
Memory (MB)	128	64	64
HardDisk	9 GB	10 GB	700 MB

protocol. It sends normal RTSP/HTTP requests to the web server, receives server responses and then according to the response code either downloads the requested file (response code is '200') or re-connects to another server (response code is '302'). The request rate is 100 requests/sec. As shown in Table 9, both maximum and average download time decrease a lot through dynamic replication and redirection. Without replication, the incoming requests queue up at the server, steadily increasing client latencies. The redirection times as measured by the client at Singapore are shown in Table 10. It should be noted that the redirection time has no close relation with the file size since the redirection request only contains the request header and not the body content. Therefore, by considering this redirection cost, in our implementation, we do not allow requests for small-size objects to be redirected or only to be redirected few times.

5.2. Experiments from China

In the experiments, the size of a request file is 19 KB since we could manage only a dial-up access to the Internet. We decided to carry out the performance study even with a dial-up access since the study can still give an idea of the time taken for RTSP/HTTP redirections. The download time from three servers and redirection times are shown in Tables 11 and 12, respectively. These result data are the average of ten trials.

As shown in Table 11, the response from USA and Germany servers are much faster than Singapore server to China clients. One reason for this is that our measurement is taken in the afternoon in terms of Singapore Time Zone while this time is evening in USA and early morning in Germany, and hence the network traffic are not so heavy as in Singapore. By taking advantage of dynamic replication and redirection in our framework, the download time for clients accessing Singapore server is deduced 14–20%. However, the redirection from Germany machine to USA machine is not so beneficial. The performance

Table 9
Download time (File size: 350 KB, client at Singapore)

Download time (s)	Without redirection		With redirection	
	Maximum	Average	Maximum	Average
Video	134	61	45	38
test04	76	42	45	36
Yurong	18	17	–	–

Table 10
Redirection time (client at Singapore)

Redirection (from/to)	video/yurong	test04/yurong	test04/video
Redirection Time (s)	8	8	5

is degraded by unnecessary redirection when two servers respond in the similar speed without redirection.

The performance measurements indicate that RTSP/HTTP redirections help in improving response time in a geographically distributed web servers framework, even though the redirection times are quite significant. Also, the studies indicate that it might be useful to consider the local time of a web server before RTSP/HTTP redirection as a predictive quantity for network/server load.

6. Simulation experiments

The system we are considering consists of N distributed servers with the same capacity. A set of clients are connected to Internet through local gateways. From the point of view of the web servers, these clients are only identifiable from their domains. Initially, there is one copy of each document in the system. Five thousand documents are distributed among the servers in round robin fashion. Once the server reaches the threshold limit, it will go through all its objects, starting with those that have a higher rate of requests and attempts to replicate them. The choice of destination server for replication is using algorithm 2 in Section 3. The object assignment information will be updated among servers once object replication is completed.

The distribution of requests to objects is based on Zipf's distribution (Cunha et al., 1995; Zipf, 1949). According to Zipf, if pages are ranked according to their access frequency, then the popularity of ith most popular page is proportional to $1/i^{(1-x)}$. This distribution for a set of L items is given by $q_i = c/i^{(1-x)}$ for each $i \in \{1, \ldots, L\}$, where $c = 1/[\sum_{i=1}^{L} 1/i^{(1-x)}]$ is a normalization constant (Zipf, 1949). The inter-request time and each request service time are exponentially distributed. Here a request may consist of several hits per page. No caching of documents is considered.

A request incurs delay due to queuing and processing at servers, redirection, network bandwidth reservation. Initially, the network bandwidth from each server local network is assumed to be 40 Mbps (DS3 link), all clients are assumed to connect to Internet through 256 Kbps cables. The geographical proximity is determined by the number of hops between the client's domain (local gateway) to the server. This value is relatively static (Paxon, 1997) and can be obtained directly from the routing table without incurring any additional network load. According to Internet hop count statistics in (Guyton and Schwarts, 1995) we obtain the value from a uniform distribution in the interval (Arlitt and Williamson, 1997; Andersen et al., 1996; Manly and Seltzer, 1997; Paxon, 1997; Rabinovich and Lazowska, 1992; Sayal et al., 1998; Schwetman, 1990; Schulzrinne H; Tewari et al., 1998; Wolfon et al., 1997; Yang and Luo, 1998; Zipf, 1949; Zona Research, 1999).

Table 11
Download time (file size: 19 KB, China

Download time (s)	Without redirection	With redirection
yurong	49	39
test-04	20	21
video	15	20

The simulation system is developed using CSIM package by Schwetman (Schwetman, 1990) in C language on SUN Ultra 2 workstations. Each value is the result of five or more simulation runs with different seeds, where each run is four hours.

6.1. Simulation parameters

Table 13 summarizes the parameters used for simulation. The simulation variables include:

1. request arrival rate
2. Zipf parameter for varying client access patterns
3. replication threshold h, which controls when to perform dynamic replication
4. server status update interval

We carried out a series of simulations to evaluate the effects of each combination of the above variables on the overall performance of the system. In order to reduce the complexity of the simulation, during each simulation we only vary one variable while keeping the others intact. The default values used for the variables are shown in Table 14.

6.2. Simulation metrics

We measure the following metrics in our experiments:

1. average waiting time per request, which is the duration from requests arriving to being served.
2. maximum waiting time, which is the maximum time a client has to wait during the simulation period
3. system imbalance rate, which is measured as a ratio of the maximum of requests served by a server to the mean computed over the total simulation period.
4. number of generated replicas.

Table 12
Redirection time (file size: 19 KB, China)

Redirection (from/to)	yurong/video	yurong/test04	test04/video
Redirection time (s)	10	10	6

Table 13
Simulation parameters

Parameters	Values
Request arrival rate	100–700 requests per hour
Request selection distribution	Zipf Distribution (Skew Value 0.0–1.0)
Number of servers	8
Number of domains	50
Request service time	exponential (mean 45 min)
Number of documents	5000
Server capacity	100 requests per second
Network bandwidth	40 Mbps
Replication threshold	0.1–0.9 of Server capacity (10–90 reqs per second)
Server status update interval	50–2000 s

It should be observed here that we do not consider any caching mechanisms during the simulation as the intention is to examine only the parameters influencing the object-level scalable web framework. As observed earlier (Section 2), we consider caching as complimentary to the object-level scalable web framework. The framework processes the requests that are missed in cache.

6.3. Simulation experiments

Fig. 6 a histogram of average waiting time during one simulation run (4 h) under the request rate of 800 requests hourly. It demonstrates that the system successfully distributes load and removes hot spots from network by dynamic replication. Without replication, requests are queued up at the popular servers, steadily increasing client latencies.

6.3.1. Effect of request arrival rate

Fig. 7 the average waiting time before a request is served between the case with dynamic replication and without replication. Assume no client cancels a request once it joins the queue. It shows that server workload based routing policy keeps give waiting time within 8 s. Zona's Research's (Zona Research, 1999) eight-second rule states: if a Web page does not load within eight seconds, the customer will abandon the request. The routing policy of server redirection based on geographical factor is effective when request rate is low and it exhibits poorly when request rate increases. This is because the network topology often does not correspond to geographic proximity and this policy is not

Table 14
Default values for simulation variables

Variables	Default Values
Request selection distribution	Zipf distribution(Skew Value 0.5)
Request arrival rate	800 requests per hour
Replication threshold	0.4
Server status update interval	300 s

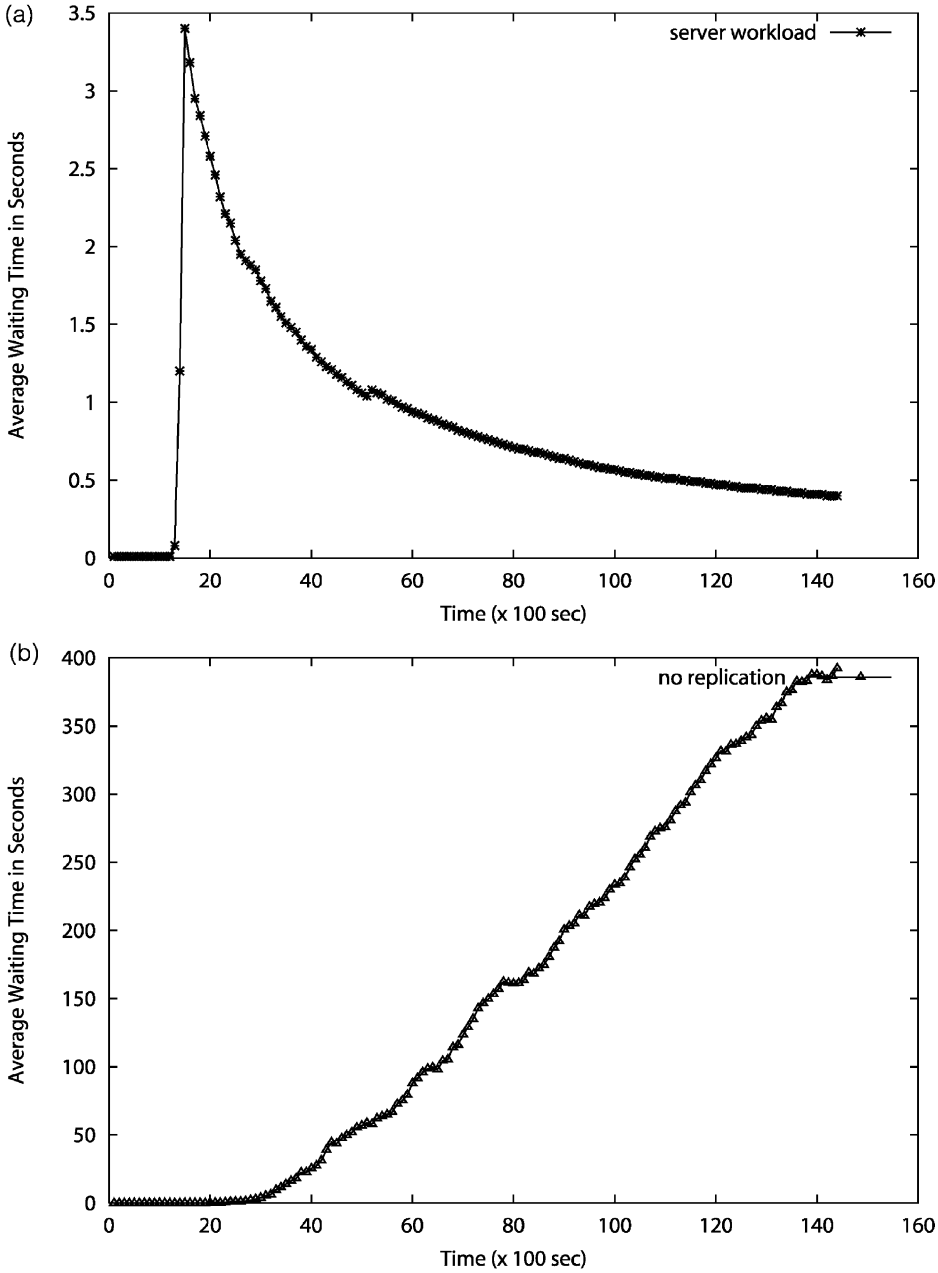


Fig. 6. Histogram of average waiting time (a) dynamic replication; (b) no replication.

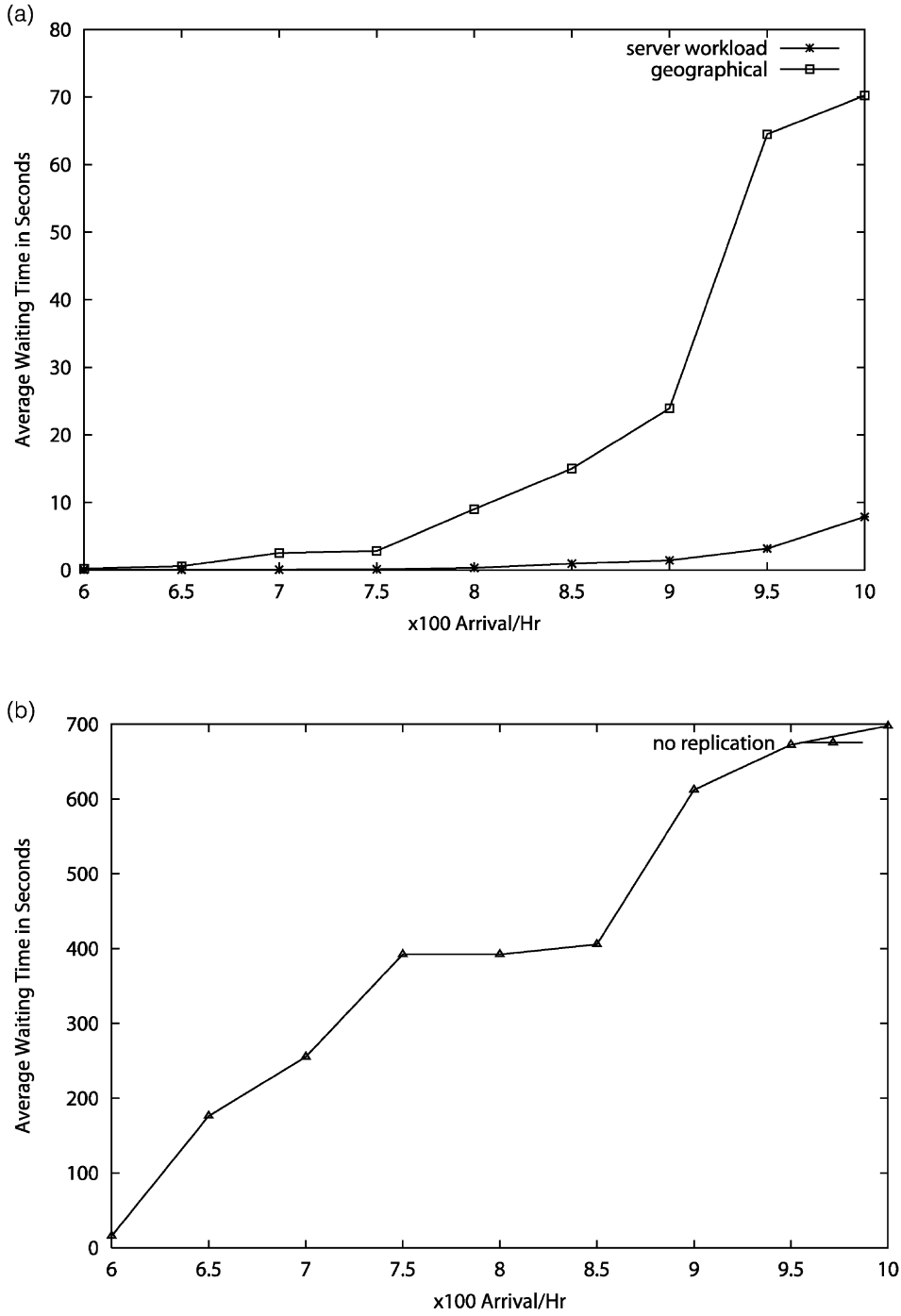


Fig. 7. Effect of request arrival rate on average waiting time.

responsive to the server workload. When no replication is done, there is only a single copy of each movie in the system during whole simulation period. Hence the performance is very bad with the average waiting time reaching nearly 700 s with request rate of 1000 requests per hour. Fig. 8 shows the maximum waiting time.

Fig. 9(a) plots the effect on system imbalance rate. It shows that workloads become more balanced among servers through dynamic replication, while the workload is not able to be distributed among servers in the case of no replication. Fig.9(b) shows the relation between the number of requests and the number of new replicas. With the workload increasing(request rate increasing), more replicas are generated to distribute the workload. A maximum of 64 replicas are dynamically generated. This is very low considering the request rate is 1000 request per and the mean service time is 45 min.

6.3.2. Effect of object popularity distribution

Figs. 10 and 11(a) show how the skewness of movie popularity affects the average waiting time and maximum waiting time. The request rate is 800 requests per hour. The robustness to request distribution is important because in a real WWW environment the client accessing scenarios tend to change frequently. When $x = 1$, all movies are equally popular. In this case, the difference between system performance with dynamic replication and without replication is small. The distinction is more prominent when x is low to moderate, i.e. the skewness in the movie popularity is high. When $x = 0.1$, workload-based redirection with dynamic replication gives average waiting time of 1.61 s while no replication gives 205.09 s. Fig. 11(b) shows the workload imbalance rate as a function of the skewness of movie popularity. In all routing policies, the workload becomes more balanced when the movies become equally popular. Recall that in the initial phase, the movies are assigned to servers in a round-robin manner.

6.3.3. Effect of replication threshold

Choosing the replication threshold presents a tradeoff between dedicating more system resources for replication in order to put a new replica into service (hoping these new replicas can satisfy more incoming requests) and dedicating these resources for normal processing of arriving requests. As the threshold decreases from 0.9 to 0.1, more replicas are created (Fig. 12(b)). However, this does not lead to better performance. We observed that in geographical-based routing the performance degrades a little bit. This is because in this policy the request is always redirected to the nearest server and not responsive to the server workload. So some newly generated replicas may be wasted. In the case of server workload based redirection, more replicas lead to improved performance first (Fig. 12(a)). However, at low threshold limit the performance starts degrading with decreasing threshold limit implying that more replicas do not always lead to better performance. One reason is that too many resources are used for unnecessary replication because of the low threshold and hence the system has only few resources for the incoming requests and requests get queued for longer time at highly loaded servers.

6.3.4. Effect of server status update interval

The server status is exchanged among servers for two purposes: one is that incoming requests can be appropriately reassigned; the other is that in dynamic replication procedure

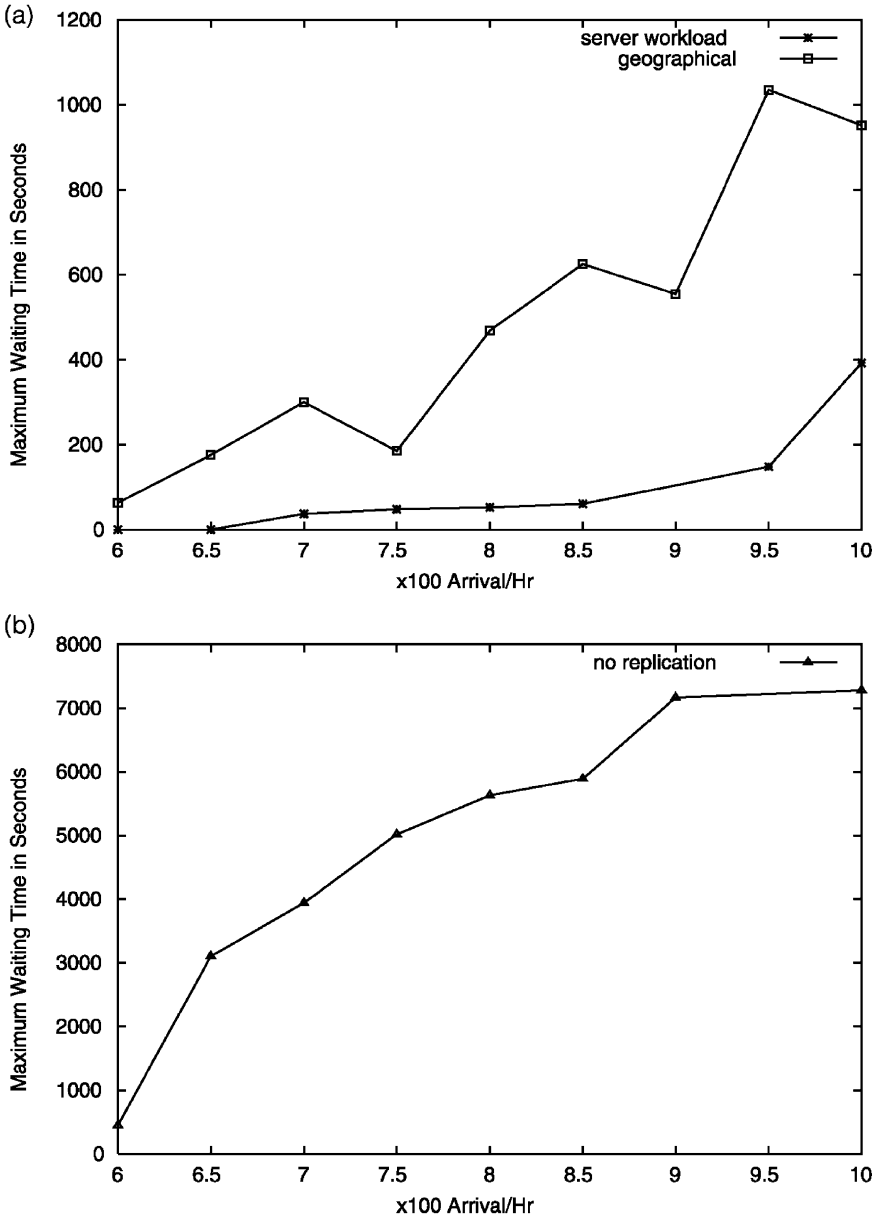


Fig. 8. Effect of request arrival on maximum waiting time.

the source server and destination server can be correctly chosen according to the server workload. Now, we consider how it is possible to minimize the overheads of the system communications and computations. Indeed, if update interval is too short, the policies can cause high computation and communication overheads due to gathering status information

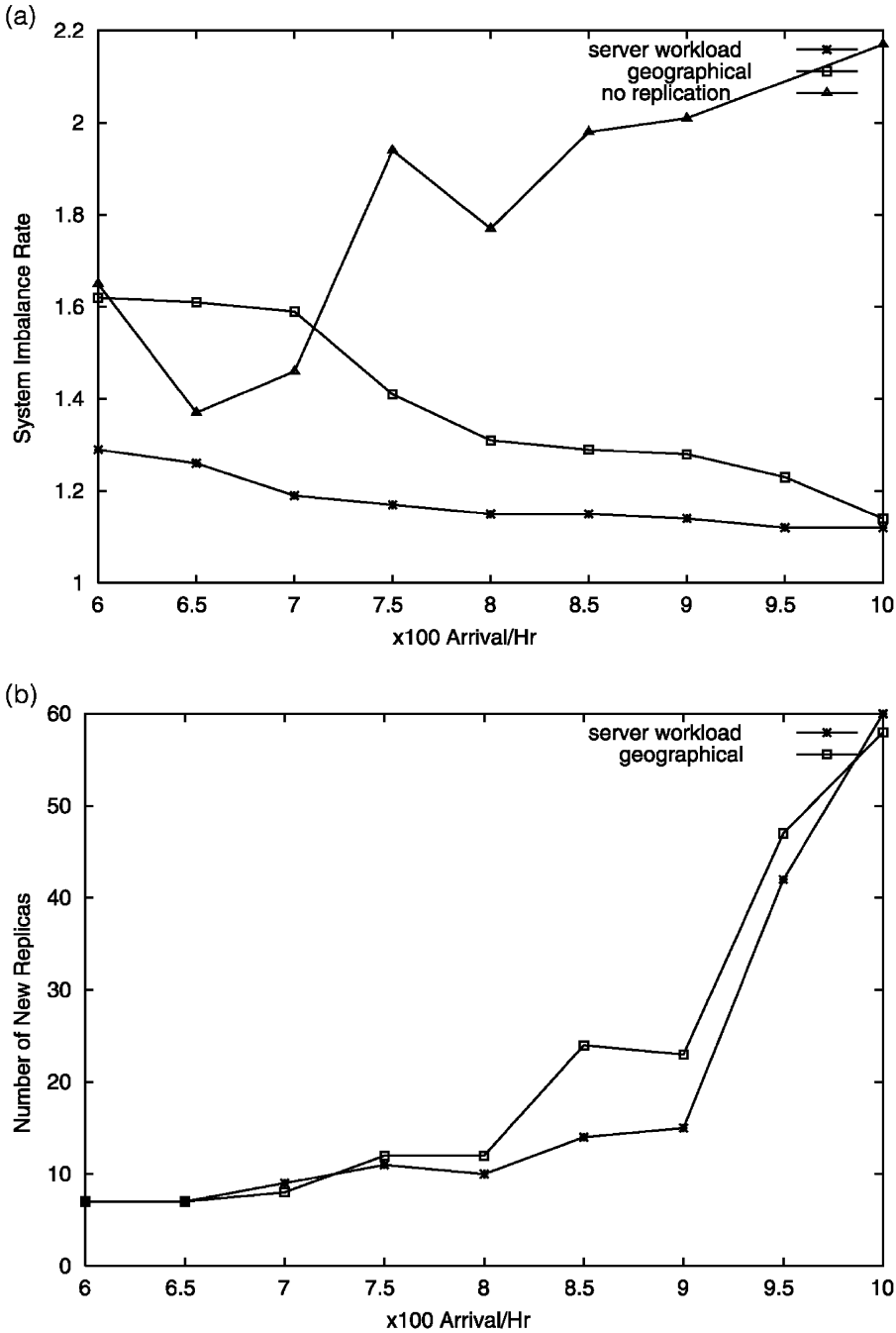


Fig. 9. Effect of request arrival rate on (a) system imbalane (b) disk consumption.

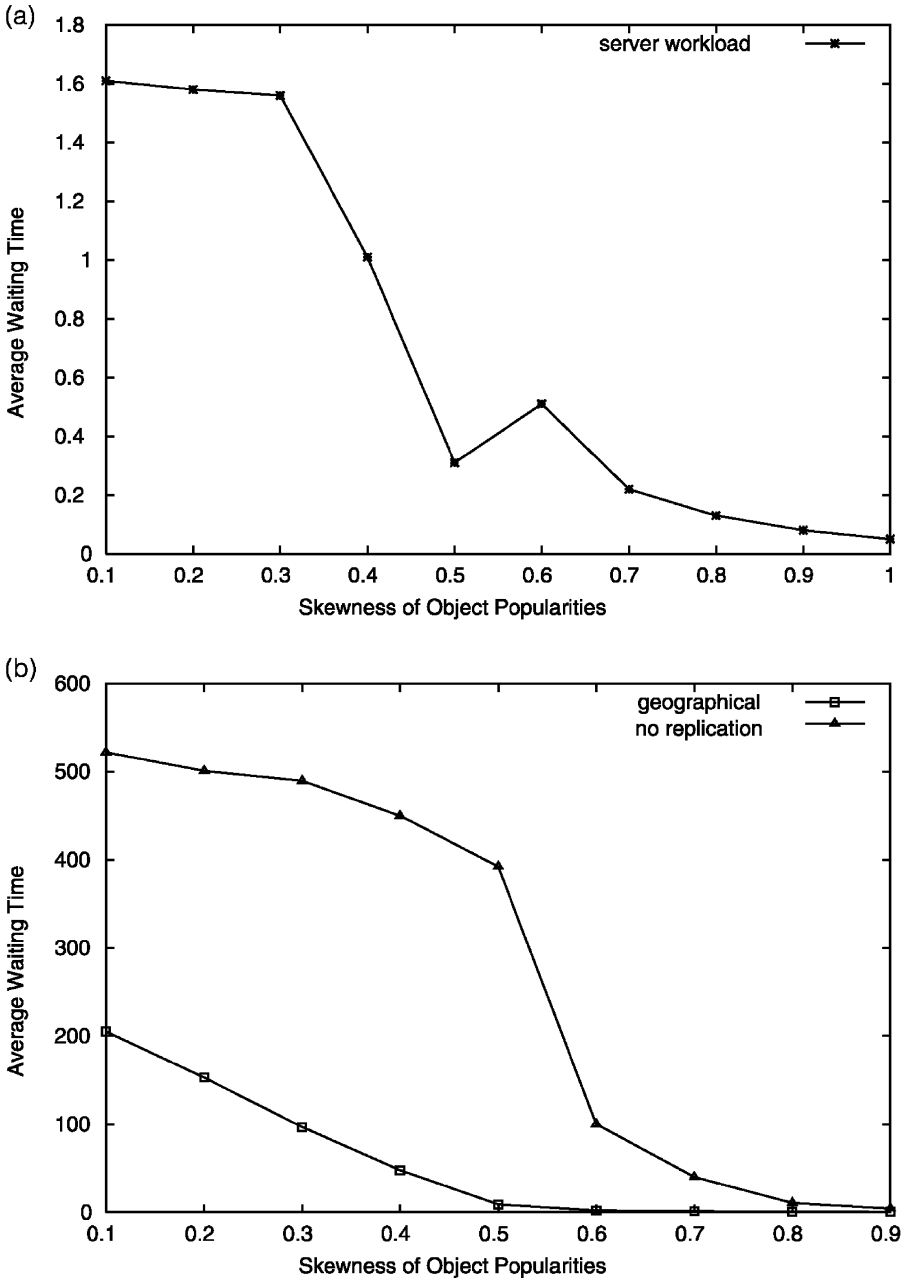


Fig. 10. Effect of object popularity distribution on average waiting time.

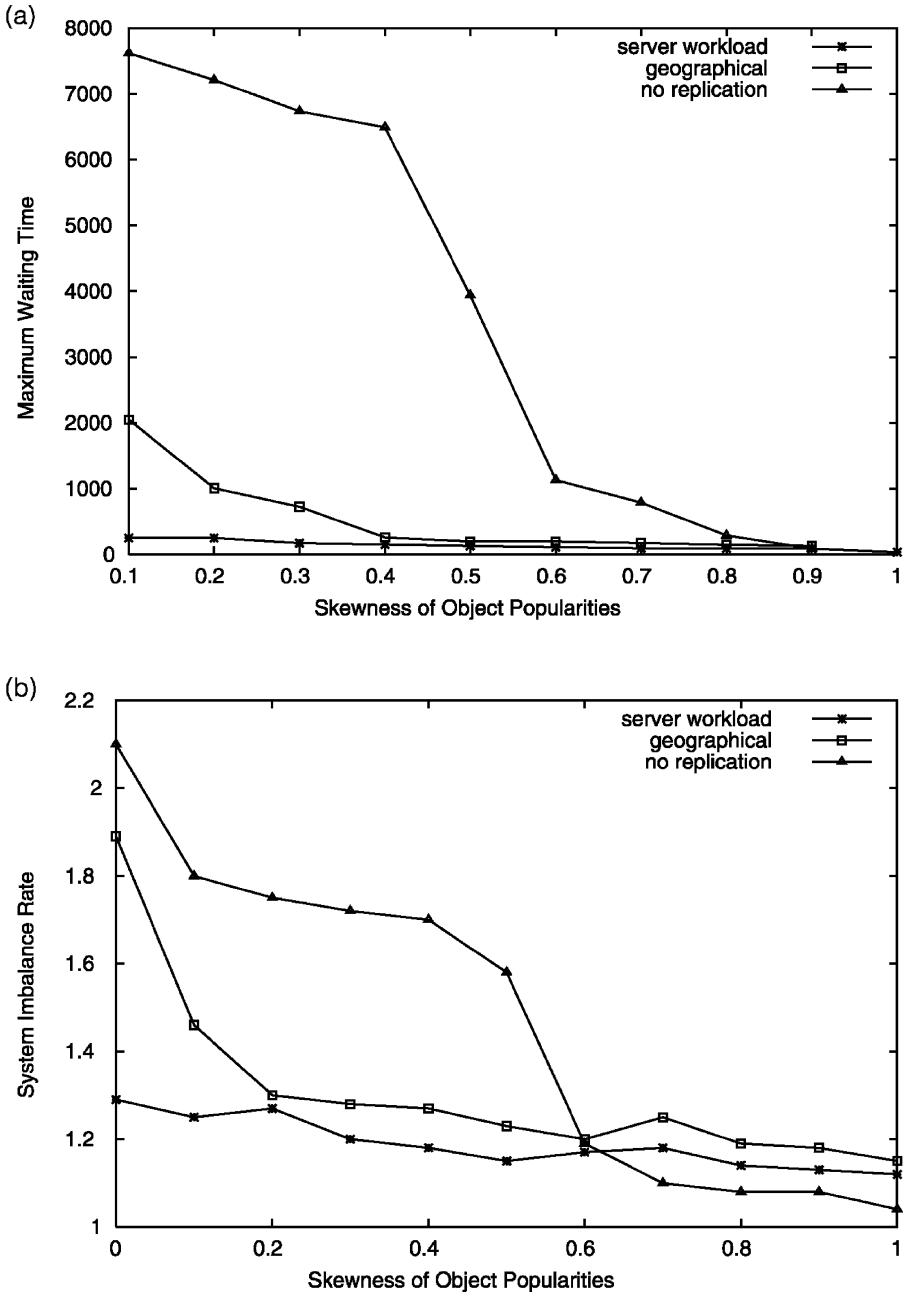


Fig. 11. Effect of object popularity distribution on (a) max waiting time (b) system imbalance.

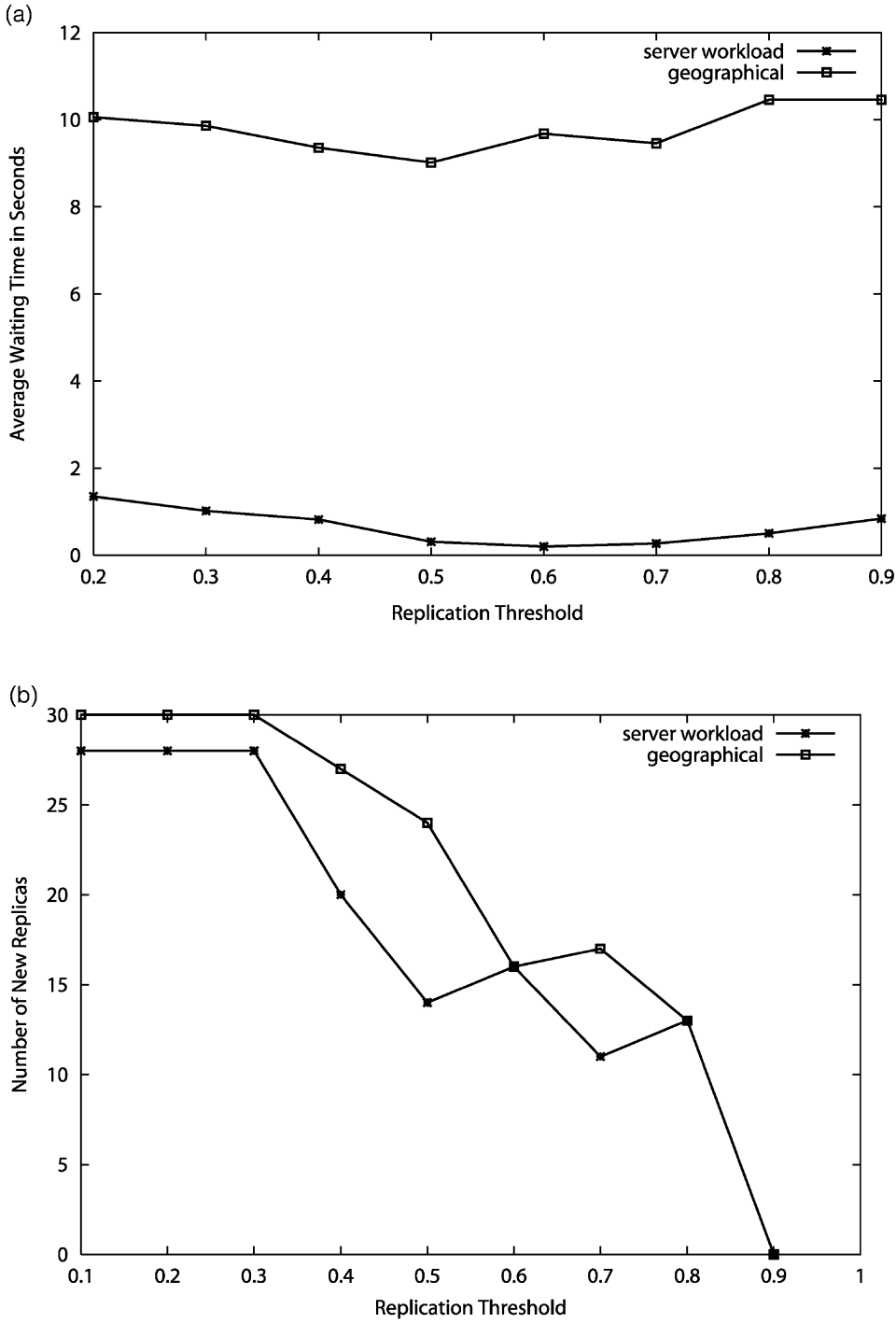


Fig. 12. Effect of varying replication threshold: (a) average waiting time (b) disk consumption.

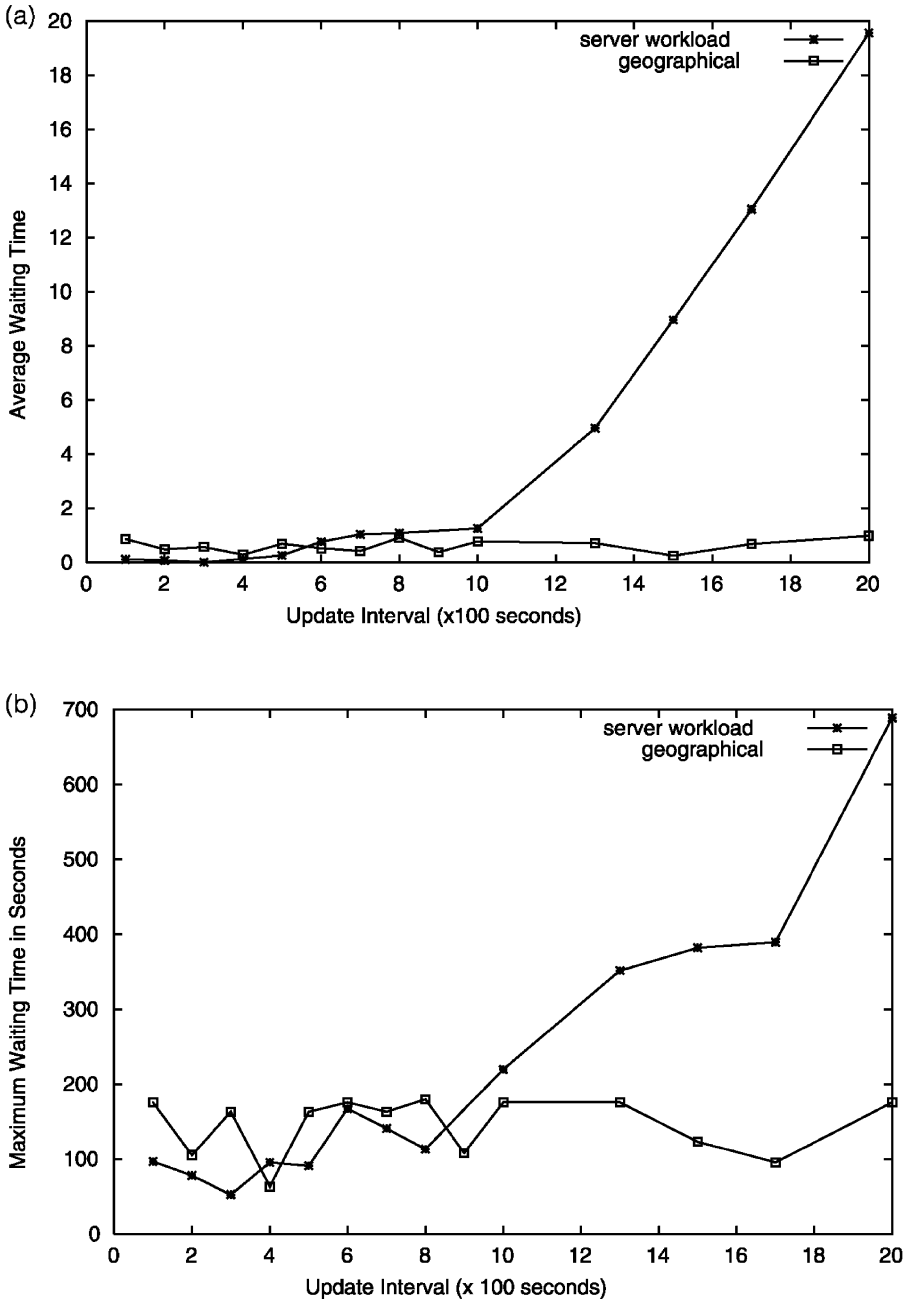


Fig. 13. Effect of varying server status update interval: (a) average waiting time (b) maximum waiting time.

and multicasting this to other servers. Hence the main goal here is to reduce the frequency of server status updating. We increase the update interval from 100 to 2000 s. The request rate in this experiment is 650 requests per hour. In Fig. 13(a) and (b) we compare the sensitivity to the update interval under server workload-based server redirection policies and geographical proximity based redirection. Server workload-based server redirection experiences big performance degradation (the average waiting time increases from less than 1 to 22 s), while the geographical-based redirection is quite insensitive to the server status update interval, keeping response within 1 s. This is because the latter always redirects based on geographical proximity and this information is quite static.

7. Summary

In this paper, we presented an object-level scalable web framework for handling large multimedia objects such as digitized movies or multimedia lectures. This framework automatically monitors access patterns, replicates, and maintains large multimedia objects among the servers without the need of full URL replication. The framework can support both RTSP and HTTP for delivering the requested objects. We employed a *traceable RTSP/HTTP redirection* approach to forward requests among servers.

We implemented the object-level scalable web framework and the whole system is an off-the-shelf tool, which does not require modification of homepages, client browsers, intermediate proxy servers or system kernels. Since the tool is developed in a modular fashion, it can be used as a platform to test various routing schemes and decision-making algorithms. This scalable approach can also be integrated with existing web servers. We installed three servers: one in the Technical University of Darmstadt, Germany, the second in the Virginia Polytechnic University USA, and the third in the National University of Singapore. We carried out performance measurements based on the implemented framework and also carried out simulation studies to evaluate this framework.

The performance measurements indicate that RTSP/HTTP redirections help in improving response time in a geographically distributed web servers framework, even though the redirection times are quite significant. Hence, RTSP/HTTP redirections over a wide area network can be quite useful for large objects such as movies or lectures (especially considering the fact that IP rewriting or TCP splicing is not efficient in wide area networks). Also, the studies indicate that it might be useful to consider the local time of a web server before RTSP/HTTP redirection as a predictive quantity for network/server load. The simulation results show that the object-level scalable web framework successfully distributes load and removes hot spots from network through dynamic replication.

Acknowledgements

We thank Prof Ralf Steinmetz, Technical University of Darmstadt, Germany, for providing us with access to the *test04* Sun machine. We thank Prof Edward A. Fox, Virginia Polytechnic University, USA, for helping us with access to the *video* Sun machine.

References

Fast Internet content delivery with free-flow, Akamai's internal technical report.

Arlitt MF, Williamson CL. Internet web servers: workload characterization and performance implications. IEEE/ACM Trans Network 1997;5(5).

Andersen D, Yang T, Holmedahl V, Lbarra OH. SWEB: toward a scalable world wide web server on multicomputers. Proc of IPPS'96, Honolulu 1996;April.

Cardellini V, Colajanni M, Yu PS. Redirection algorithms for load sharing in distributed web-server systems. Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, Los Alamitos, CA: IEEE Computer Society Press; 1999.

Colajanni M, Yu PS, Dias DM. Analysis of task assignment policies in scalable distributed web-server systems. IEEE Trans Parallel Distrib Syst 1998;9(6).

Cunha CA, et al. Characteristics of WWW client-based traces. Technical Report TR-95-0101. Department of Computer Science, Boston University; April 1995.

Dias DM, et al. A scalable and highly available web server. Proc Fourth IEEE Int Conf (COMPCON'96), Technol Information Superhighway; 1996. p. 85–92.

Fei Z, Bhattacharjee S, Zegura EW, Ammar MH. A novel server selection technique for improving the response time of a replicated service. Proc INFOCOM98 1998.

Fielding R, Gettys J, Mogual J, Frystyk H, Berners-Lee T. Hytertext Transfer Protocol—HTTP/1.1. RFC 2616 1999;June.

Fox A, Gribble S, Chawathe Y. Cluster-based scalable network services. Proc SOSP'97, France, October, 1997. p. 78–91.

Gwertzman J, Seltzer M. The case for geographical push-caching. Proc HotOS Workshop; 1994.

Guyton JD, Schwarts MF. Locating nearby copies of replicated Internet servers. Proc SIGCOMM; 1995.

Hunt GDH, Goldszmidt GS, King RP, Mukherjee R. Network dispatcher: a connection router for scalable Internet services. Proc WWW7, Brisbane, April, 1998.

Jajodia S, Mutchler D. Dynamic voting algorithms for maintaining the consistency of a replicated database. ACM Trans Database Syst 1990;230–80.

Kwan TT, McGrath R, Reed DA. NCSA's World Wide Web Server: Design and Performance. Computer 1995; 28(11):68–74.

Long DDE, et al. Regeneration protocols for replicated objects. Proc Fifth IEEE Int Conf Data Engng 1989.

Manly S, Seltzer M. Web facts and fantasy. USENIX Symp Internet Technol Syst 1997.

Paxon V. End-to-end routing behavior in the Internet. ACM/IEEE Trans Networking 1997;5(5).

Rabinovich M, Lazowska ED. Improving fault-tolerance and supporting partial writes in structured coterie protocols for replicated objects. Proc ACM SIGMOD Int Conf Mgmt Data 1992;226–35.

Sayal M, et al. Selection algorithms for replicated web servers. Workshop on internet server performance 1998; June.

Schwetman H. CSIM User's Guide. MCC Technical Report No. ACT_126_90. Microeletronics and Computer Technology Corporation, Austin, TX; March 1990.

Schulzrinne H. World Wide Web: whence, whither, what next? IEEE Network; 1996. p. 10–17.

Tewari R, Dahlin M, Vin HM, Kay JS. Beyond hierarchies: design considerations for distributed caching on the Internet. Technical Report TR98-04. Department of Computer Sciences, University of Texas, Austin; 1998.

Wolfon O, Jajodia S, Huang Y. An adaptive data replication algorithm. ACM Trans Database Syst 1997;June.

Yang CS, Luo MY. Design and implementation of an environment for building scalable and highly available web server. Proc 1998 Int Symp Internet Technol 1998;April:124–31.

Zipf GK. Human behaviour and the principles of least effort. Cambridge, MA: Addison-Wesley; 1949.

Zona Research. quoted in Interactive Week; September 6, 1999.