Cyclone: A High-Performance Cluster-Based Web Server with Socket Cloning

Yiu-Fai Sit, Cho-Li Wang, Francis Lau
Department of Computer Science and Information Systems
The University of Hong Kong
E-mail: {yfsit, clwang, fcmlau}@csis.hku.hk

Abstract

With the ever-growing web traffic, cluster-based web server is becoming more and more important to the Internet's infrastructure. Making the best use of all the available resources in the cluster to achieve high performance is thus a significant research issue. In this paper we introduce Cyclone, a cluster-based web server that can achieve nearly optimal throughput. Cyclone makes use of a novel network support mechanism called Socket Cloning (SC) together with the concept of hot object replication to obtain high performance. SC allows an opened socket to be moved efficiently between cluster nodes. With SC, the processing of HTTP requests can be migrated to the node that has a cached copy of the requested document, thus bypassing any cache transfer between cluster nodes. To achieve better load balancing, frequently accessed documents (hot objects) are replicated to other cluster nodes by forwarding cached copies. Trace-driven benchmark test using http_load shows that Cyclone outperforms existing approaches and can achieve a throughput of 14575 requests/s (89.5 MBytes/s), which is 98% efficiency of the available network bandwidth, with 8 web server nodes.

1 Introduction

Exploiting the power of commodity-off-the-shelf components to achieve high performance has long been one of the main goals of cluster computing. Many applications have been designed and built in cluster environments with the hope to achieve this goal. Networks of Workstations [3] and Beowulf [43] type clusters are deployed in many research institutes and government agencies for high performance parallel computing. Interactive rendering systems like WireGL [31] and computer aided film production also rely on clustering to produce the required frames within reasonable time. Among all these applications, cluster-based web server is the most popular. Many web sites are now equipped with "web server farm" to cope with the huge demand of the visitors from all over the world.

In a cluster-based web server, all the machines are connected together and act as one single server to the outside world. The aim is to fully utilize the resources within the cluster to obtain high performance and scalability. In an ideal setting, web site administrators can add more computers to the cluster to increase the server's capacity. Cluster-based web servers have become commonplace in large web sites and with the ever-growing web traffic, they will become more and more important to the Internet's infrastructure.

The aim has not been met, however. Lots of research effort has been putting into this area for nearly a decade to achieve high performance and scalability. Many new technologies have been evolved, but it appears that no system reported in the literature can fully utilize a cluster's resources to achieve the maximum possible throughput of the cluster. In fact, there are several challenges that we have to overcome simultaneously to accomplish this goal. They are: efficient request distribution, fine-grained

load balancing, effective memory utilization, optimized processing of popular objects, and minimal network resource consumption in coordinating the web server nodes.

- **Efficient Request Distribution:** A cluster-based web server can be composed of thousands of nodes. There have to have a mechanism for the system to decide efficiently which node should handle a particular request to fully utilize the available servers. Such mechanism is called request distribution. There are three different approaches for this. In the Web-Client Scheduling approach [36] and DNS approach [13], the web client will first find out which node it should contact before sending out the request. This is a one-time cost because the client will use the same node for subsequent requests. In the Dispatcher approach, a router called dispatcher located in front of the web server nodes acts as the entrance to the cluster. All the request distribution decisions are made at the dispatcher and the cost is incurred in every HTTP session. Request distribution has to be as efficient as possible because it affects the client's perceived performance of the cluster. This is especially important to the Dispatcher approach because the overhead in the dispatcher limits the maximum number of requests that the cluster can accept. In some distribution mechanisms like LARD [38], the overhead is so large that the dispatcher can handle at most ten web server nodes only.

- **Fine-Grained Load Balancing:** In persistent HTTP [24], a single TCP connection can contain many HTTP requests. Traditional coarse-grained load balancing approaches, e.g. Layer-4 dispatching, that work at the granularity of connection are thus insufficient. A node that is idle when the first request arrives can become heavily loaded when the later requests are sent to it through the same TCP connection. Because of the connection-oriented nature of TCP, these requests have to be handled by this loaded server. Load balancing at the granularity of HTTP request is therefore necessary to achieve high performance.

- **Effective Memory Utilization:** While clustering can provide a virtually unlimited aggregated memory size, the memory is physically fragmented across the nodes. Most cluster-based web servers depend solely on the file cache in each node to avoid expensive hard disk accesses. There is no cooperation of the caches in the nodes. This will lead to poor performance because the effective memory size of the cluster for caching the documents equals to the available cache memory of a single node only. Maximizing the effective cache size thus plays an important role in achieving high performance.

- **Optimized Processing of Popular Objects:** One of the characteristics of web server workload is concentration of references [6, 7]. There is usually a small group of documents or objects that are frequently accessed in a web server. Poor performance will result if these documents have to be retrieved from the hard disk every time when they are requested. The system should be able to determine which document is popular and keep it in the memory cache as long as it remains 'hot'. In contrary to maximizing the effective cache size, these popular documents should be stored in more than one of the caches. Document partitioning strategies that do not consider this will lead to hot spot in the cluster.

- **Minimal Network Resource Consumption:** Since each of the web server nodes in the cluster is an autonomous machine, messages have to be exchanged among them for coordination and

performance enhancement. These messages range from short status or load queries to large cache transfers. While message exchange is necessary for the normal operation of the cluster, they have to be kept to minimum because these messages compete with the outgoing HTTP response for the same network resource. Techniques like cooperative caching [21] that try to achieve high performance by using network resources to avoid expensive disk accesses do not address this issue and therefore the achievable throughput is limited.

As we have seen, failing to overcome any of these challenges will lead to poor performance and scalability of the cluster. They have to be solved simultaneously in order to fully exploit the potential of the system. However, some of these challenges can be in conflict with each other: replication of hot objects reduces the effective cache size of the cluster, and fine-grained load balancing is usually achieved at the expense of higher request distribution overhead. Addressing all these challenges with a balanced solution is an important issue in achieving high performance for cluster-based web server.

Previous work in this area only concentrated in some of these challenges and left out the others in their considerations. In this paper we introduce Cyclone, a cluster-based web server that addresses all the mentioned challenges simultaneously by *Socket Cloning (SC)* [41] and the concept of hot object replication [16].

Socket Cloning allows an opened socket to be moved efficiently between cluster nodes. This allows fine-grained load balancing with low overhead. With SC, the processing of HTTP requests can be migrated to the node that has a cached copy of the requested document, thus bypassing any cache transfer between cluster nodes. As a result, the effective cache size is increased and network resource consumption is minimal. To further minimize the overhead in processing an HTTP request, frequently accessed documents (hot objects) are replicated to other cluster nodes by forwarding cached copies. Trace-driven benchmark tests show that Cyclone outperforms existing approaches and can achieve a throughput of 89.5 MBytes/s at 14575 requests/s, which is 98% of the total available TCP bandwidth, with eight web server nodes.

The rest of this paper is organized as follows. In the next section, we present a detailed background on the work of cluster-based web server, their advantages, and disadvantages with respect to the challenges discussed. In Section 3, we present the architecture of Cyclone and its two enabling mechanisms: Socket Cloning and hot object replication. Benchmarks on Socket Cloning and Cyclone are reported and analyzed in Section 4 and 5 respectively. We discuss related work in Section 6. Finally we conclude and give some pointers for future work in Section 7.

2 Background

A pool of autonomous server nodes that are tied together to act as a single web server to the outside world is called a cluster-based web server. It has become a common solution to high-traffic web hosting in recent years because of its scalability and performance. Depending on the major strategy used to achieve high performance, existing cluster-based web servers systems can be divided into two main categories: dispatcher-based and server-based. In the dispatcher-based approach, client requests are distributed to the web server nodes by the dispatcher to achieve load balancing and high throughput. On the other hand, server-side approach uses some special caching mechanism at the server nodes that

can avoid excessive disk operations to improve the overall throughput. The rest of this section will give a detailed overview of these two categories, their advantages, and problems.

2.1 Dispatcher-Based Approach

Dispatcher-based approach emphasizes on using request distribution to achieve load balancing. All the packets from the clients to the cluster will first reach the dispatcher, which routes the packets to the web server nodes based on some load balancing policies. There exist many mechanisms for request distribution, which can be further divided into two categories, Layer-4 and Layer-7, based on the OSI layer at which the distribution decision is made [14, 15].

2.1.1 Layer-4 Dispatching

In Layer-4 dispatching, distribution decision is made according to the IP address and TCP port number of the client. The dispatcher does not establish connections with the clients; it only routes packets to the chosen web server node, which establishes TCP connection directly with the client. Since TCP is a connection-oriented protocol, the endpoints of a connection cannot be changed. The dispatcher thus has to route all the packets in the same connection to the same web server node.

Layer-4 dispatching can be further divided into two sub-categories, Layer-4/2 and 4/3, depending on how packets are routed to the web server nodes [28]. In Layer-4/2 dispatching systems, e.g. eNetwork Dispatcher [32], packets are forwarded to the nodes by modifying the link layer (Layer 2) destination address. All the web server nodes bind to the same virtual cluster address and they can send the replies to the clients directly without passing through the dispatcher. In Layer-4/3 dispatcher like MagicRouter [2], the destination IP address is altered to the chosen server node's address with the IP checksum updated. The dispatcher also acts as the gateway of the web server nodes. HTTP response packets are sent to the dispatcher, which changes the source address back to the cluster's address, computes the IP checksum and sends it out to the clients.

Both of the sub-categories achieve the same level of load balancing. However, Layer-4/2 dispatching has a larger routing capacity. The dispatcher does not have to handle outbound packets, which are a lot more than the inbound packets. In Layer-4/3 dispatching, the cluster's throughput is limited by the outgoing bandwidth of the dispatcher. On the other hand, in Layer-4/2 dispatching the dispatcher and the web server nodes have to be connected within the same physical network. In a cluster environment, this constraint has little impact since the nodes are likely to be connected via a high-speed local network.

Layer-4 dispatching is efficient due to its simplicity, but fine-grained load balancing is difficult to achieve. Once a TCP connection is established, the two endpoints cannot be changed. When a node becomes heavily loaded, there is no way to move existing connections to other less loaded machines. The node has to handle the requests until the connections are closed. This problem becomes more severe with persistent HTTP. Multiple requests are sent within one connection and the least loaded node can become heavily loaded later in the lifetime of the connection. With Layer-4 dispatching, connections cannot be moved to another node to balance the load.

2.1.2 Layer-7 Dispatching

Layer-7 dispatching, often called content-based distribution, looks into the HTTP request to decide which cluster node should serve a request. Since HTTP request is sent after the TCP connection is established, the dispatcher has to first set up a connection with the client before it can parse the request and made the distribution decision. After that, processing of the request is passed to the chosen web server node. There are four different mechanisms that enable Layer-7 dispatching: relaying [34, 51], TCP splicing [5, 19, 45], relaying with packet writing [49, 50], and TCP handoff [9, 10, 17, 38, 44].

Relaying: In relaying [34, 51], the dispatcher acts as the middleman between the clients and the web server nodes. Clients establish connections directly with the dispatcher and send HTTP requests to it. After parsing a request and applying some content-aware load balancing algorithms, the dispatcher sends this request to the chosen web server node via another connection. The web server node then processes the request and sends the HTTP response to the dispatcher. The received response is copied to the buffer of the connection with the client and sent out by the dispatcher. While this mechanism is simple to implement, it suffers from low performance. Copying of HTTP response between two connections causes large overhead because the size of response can be very large and it can span across many packets. The dispatcher also limits the rate that the cluster accepts requests and hence the resultant performance is reduced. In addition, the maximum throughput cannot exceed the bandwidth of the dispatcher because all the packets have to pass through it. IBM's Web Server Accelerator [34] takes a step further by performing caching at the dispatcher. When the dispatcher receives an HTTP request, it will first try to find the document in its own cache to serve the request. Relaying is only needed when it does not have the document in cache. Such technique is sometimes called reverse proxy. This can greatly improve the performance of relaying, but the mentioned limitations still applies.

TCP Splicing: TCP splicing [5, 19, 45] is an optimization of the relaying mechanism. In relaying, responses from the web servers have to be copied into the send buffer of the client's connection. TCP splicing eliminates copying of data and context switching by establishing a pipe between the input and output buffers of the two sockets involved. The two connections are virtually linked together. This can increase the forwarding efficiency, but the throughput of the cluster is still limited by the dispatcher's bandwidth.

Relaying with Packet Rewriting: This is another optimization of relaying [49, 50]. Before the cluster starts, the dispatcher establishes some connections with the web server nodes. After the dispatcher has set up a connection with the client and parses the HTTP request, it will forward the request to the chosen web server node through one of the already-established connections by rewriting the TCP and IP headers of the packet. The rewritten packet appears exactly the same as it is sent from the dispatcher. The web server node then sends back the HTTP response to the dispatcher, which again rewrites the headers of the packet and forwards it to the client. This is similar to Layer-4/3 dispatching, but the dispatcher has to establish connection with clients. Since the packets can be rewritten just above the device driver layer, they do not have to traverse up to the protocol stack as in TCP Splicing. This further reduces the time in forwarding HTTP response from the web server nodes to the client. Since this is just an optimization of relaying,

it still suffers from the same drawbacks of relaying. Figure 1 shows the mechanisms of these relaying variants.

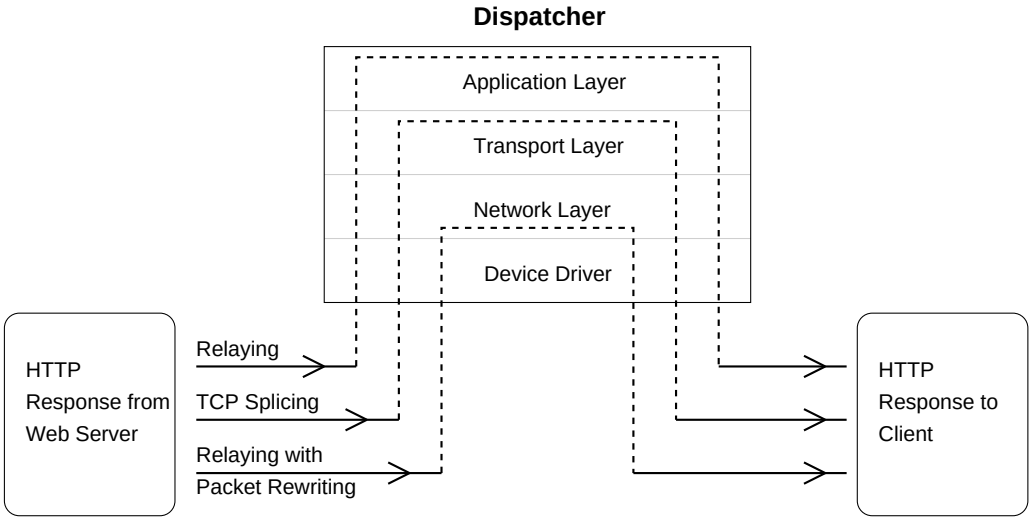

Figure 1: Mechanisms of Different Variants of Relaying

TCP Handoff: TCP handoff [9, 10, 17, 38, 44] takes another approach. After the request has been parsed by the dispatcher, the connection endpoint at the dispatcher is handed off to the chosen node, which then processes the request and sends out the reply directly to the client. This removes the dispatcher from limiting the outbound bandwidth of the cluster. However, TCP handoff has very high overhead, which limits its scalability. Experiment showed that a dispatcher can only support a maximum of ten nodes [38]. Although a solution has been proposed in [10] to share the dispatcher's load among the web server nodes to lessen the problem, this in turn imposes more work in the server nodes. The overall performance is thus still limited. To support persistent HTTP, design issues of multiple handoff are briefly discussed in [9]. The main challenge is to prevent the TCP stream from draining during the process of handoff. However, there is no solution given. The lack of multiple handoff mechanism limits TCP handoff's advantage over Layer-4 dispatching in persistent HTTP connections because only the first request in a connection can be dispatched according to its content. Subsequent requests in the same connection are still forced to be directed to the same server node, as in Layer-4 dispatching.

In all the discussed mechanisms of Layer-7 approach, every request is actually parsed twice: once in the dispatcher and once in the web server. Establishing a TCP connection with each client and parsing the HTTP request introduce large overhead, and the dispatcher can become a performance bottleneck much more easily than the Layer-4 approach. This limits the use of such techniques in more complex protocols or request formats. In addition, Layer-7 approaches are inefficient for small-size responses where the dispatching overhead can be much larger than the actual transfer time of the replies.

2.2 Server-Side Approach

In server-side approach, the idea of cooperative caching has been relied upon for achieving high performance [1, 16, 20]. Each web server node maintains a cache to store web documents. When an HTTP

request is received by a web server node, it will first look up its own cache for the requested document. If it is there, the server will send out the cache to the client. Otherwise, it will ask other nodes in the cluster to see if any one has the document in cache. A node that has the cache will reply and send it to the web server node. The received cache is then sent to the client. If the document is not in any cache, the node will retrieve it from hard disk to satisfy the client's request. A mechanism called *cache forwarding (CF)* is needed to transfer cached objects between the nodes. There are at least two messages involved: the request for a copy from a peer node, and the reply. Since network latency in modern local area network is much smaller than the latency of secondary storage, forwarding a small object through the network is still faster than performing local disk I/O. However, cooperative caching only works best for small-size transfers. Commodity SCSI and IDE hard disk has a throughput of more than 100 MBytes/s nowadays while the most popular network is still 100Mbits/s Fast Ethernet. As a result, reading a large file from local hard disk is probably more efficient than transferring a cache through the network. CF also consumes network resources in the cluster, which will limit the overall throughput.

2.3 Summary

Most of the commercial products for cluster-based web server follow the dispatcher-based approach [18, 23, 25, 37]. A web switch or load balancer sits in front of the cluster and distributes the requests to the web server nodes by Layer-4 or Layer-7 policies. Mechanisms similar to TCP splicing are usually necessary for these switches to achieve Layer-7 load balancing without modification in the web server nodes with the exception of Resonate, Inc., which offers a mechanism that is similar to TCP handoff [39]. All these commercial products share the same advantages and drawbacks of dispatcher-based approaches, however.

Table 1 summarizes existing mechanisms to achieve higher performance in cluster-based web servers.

3 Cyclone: A High-Performance Cluster-Based Web Server

It can be seen from the previous section that current cluster-based web server systems still suffer from many drawbacks that limit their performance. In this section, we present the architecture of Cyclone, a high-performance cluster-based web server that resolves the stated problems by a novel mechanism called *Socket Cloning (SC)* [41] and the concept of hot object replication.

Socket Cloning is an efficient network support mechanism that enables an opened socket to move to another machine for communication and computation on its behalf. When a web server node decides not to handle a particular HTTP request by some load balancing policies, it can clone the socket that corresponds to the request to a more suitable node in the cluster. For example, instead of performing local disk access, a node can clone the socket to a node that has the cache copy of the requested document to serve a request. Efficient memory utilization is thus achieved. Cache forwarding is also avoided by cloning a socket. This reduces network resource consumption. Moreover, SC supports multiple cloning to provide fine-grained load balancing.

In addition to SC, Cyclone also relies on the concept of hot object replication to achieve high performance. Frequently accessed documents, called hot objects, can be cached in several nodes. Requests of hot objects can be served directly by any node that has a cached copy, instead of cloning

	Dispatcher -Based				Server-Side
Mechanism	**Layer-4**		**Layer-7**		**Cooperative Caching**
Sub-category	Layer-4/2	Layer-4/3	Relaying and its variants	TCP Handoff	-
Examples	eNetwork Dispatcher [32], LSMAC [28, 27, 26], ONE-IP [22], SASHA [29], stateless DPR [12, 11]	LSNAT [42], MagicRouter [2]	HACC [51], IBM Web Server Accelerator [34], L5 [5], [49], and [50]	LARD [38], Resonate Central Dispatch [39], WARD [17]	p-Jigsaw [16], WhizzBee [48], and [20]
Advantages	Simple, Low Overhead, High scalability.		Independent of the back-end web server, Simple to implement, Can act as a proxy.	Outbound packets do not have to pass through the dispatcher.	Reduced disk access, Larger total effective cache size.
Disadvantages	Coarse-grained load balancing, Content blind, No data partitioning, Every server node has to handle the whole set of web documents.		Connection setup rate limited by dispatcher, Throughput limited by dispatcher, High overhead, Low scalability.	Connection setup rate limited by dispatcher, Very high overhead, Low scalability, Poor support in persistent HTTP.	Consumes network resources, Does not work well with large files, Overhead in maintaining cache tables and Dispatcher -Based consistency.

Table 1: Summary of Strategies in Cluster-Based Web Servers

the socket to one particular node. This can greatly improve the performance of serving hot objects. Less frequently accessed files are still served by Socket Cloning, which is more efficient than existing mechanisms. To minimize the impact of cache replication on the cluster's effective cache size, we impose a limit on the total memory size for replication.

There is no limitation on which dispatching mechanism to be used in Cyclone. Any mechanism that is efficient enough can be deployed to distribute the request. The combination of this with Socket Cloning and hot object replication is thus a high performance approach to cluster-based web server that addresses all the problems stated in the Introduction simultaneously.

3.1 Socket Cloning

In this subsection, we will discuss the working mechanism of Socket Cloning and its use in Cyclone.

3.1.1 System Architecture

The architecture of Socket Cloning aims at providing an efficient and transparent network support system for cluster-based web servers with the discussed problems resolved. For the simplicity of the discussion, it is assumed that a Layer-4/2 dispatcher is used to distribute requests to the web server nodes in the cluster. In fact, any other lightweight dispatching mechanism can also be used.

There are three components in Socket Cloning: *SC Client, SC Server,* and *Packet Router.* The architecture is shown in Figure 2.

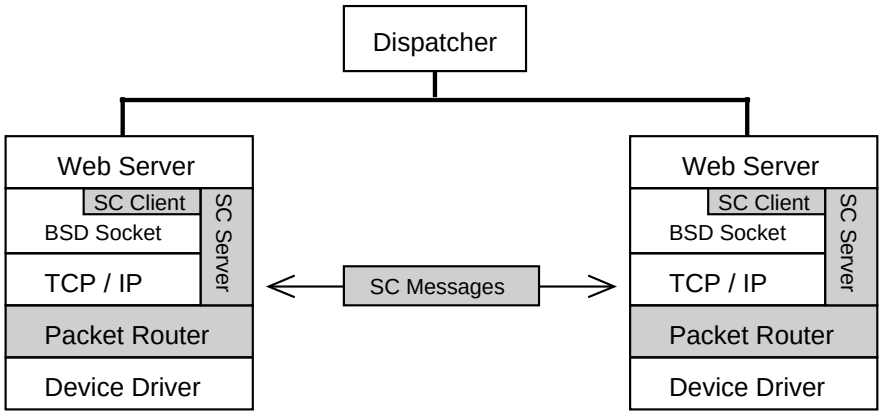

Figure 2: System Architecture of SC

SC Client provides a system call interface to the web server software in the node. When a web server decides to have another node to handle the request, it issues the system call provided by SC Client to clone the socket. SC Client then packs all the relevant information of the opened socket and sends it out to the SC Server in the remote node through a persistent connection. The whole message is called *SC Message.* When the cloning system call returns, the web server treats the request as served and processes the next request.

After receiving an SC Message, the SC Server will create a socket called *clone.* The states of the clone and the protocol stack are then reconstructed according to the information in the SC Message. HTTP requests received by the original socket are also placed in the clone's buffer. The web server software in this node will then treat the clone as an ordinary socket and process the requests in the

clone's buffer. The clone is native to this node and subsequent packets will go through normal network protocol stack. There is no extra overhead in processing the packets of a clone. Outgoing packets of the clone are sent directly to the client. Upon a successful cloning, the SC Server will send an acknowledgement back to the SC Client. It will then inform the Packet Router to route subsequent packets for that socket to the clone's node. During the execution, packets from the client will first reach the original node and be routed to the clone's node while packets to the client are sent directly from the clone's node. A triangular routing path is established. Furthermore, packets that contain non-zero TCP payload are passed to the network stack of the original node as well as routed to the clone. This ensures the clone will send the correct acknowledgement to the client.

After cloning, the original socket remains in its node. It will not be destroyed until the connection is closed. The original socket prevents the TCP stream from draining in SC. When messages are received during cloning, the original socket will receive them and copy it in a buffer. After the socket is cloned, the buffer is forwarded to it. In this way, the TCP stream is prevented from losing messages. Figure 3 shows how a series of HTTP requests are handled in the system.

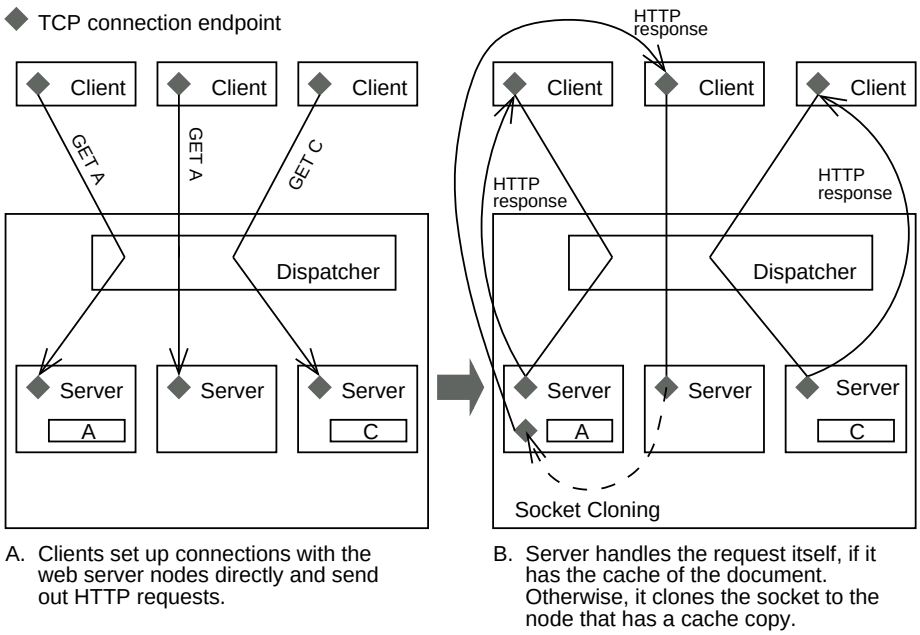

Figure 3: SC in Cluster-Based Web Server

For comparison with other content-aware approaches, Figure 4 and Figure 5 show the logical flow of how the same series of HTTP requests are processed in relaying and TCP handoff respectively. In both cases, the client first has to set up a connection with the dispatcher, which then parses the requests one by one. After that, the connection is relayed or handed off to the chosen web server. This sequential request processing and heavyweight connection handoff or relaying impose a great limit in cluster-based web server. In SC, requests are distributed by a Layer-4 dispatcher (or other lightweight mechanisms, such as DNS approach [13]), which has very small overhead. Clients set up connection with the web server nodes directly and the processing of different requests can thus be carried out in parallel.

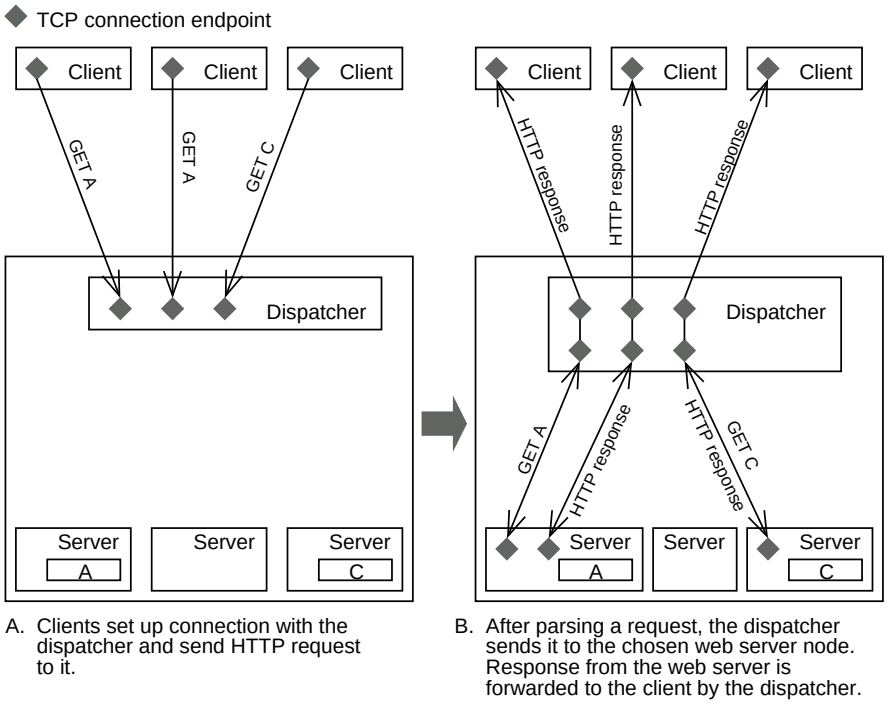

Figure 4: Relaying and Its Variants

3.1.2 Triangular Implicit State Synchronization

The clone and the original socket have the same state right after cloning. However, the states of the clone and the original socket can become different when the clone has sent out some packets. As the original node will process further requests from the client, the states of the original and cloned socket have to be synchronized. Packets sent from an unsynchronized socket will appear erratic to the client and packet from the client will be treated as an abnormal one in such socket. The states of the sockets thus have to be consistent before the original socket can take over control of the connection again to serve subsequent requests in a connection.

Because of triangular routing, the original socket does not know what the clone has sent. As a result, some mechanism is needed to update the states of the original socket to keep the states consistent. This usually involves inter-node communications and may adversely affect scalability of the whole system. In our system, explicit synchronization is not required. The Packet Router takes full advantage of the acknowledgement received from the client to manipulate the state of the original socket so that the states become synchronized after the acknowledgment is received at the clone's node. Such implicit synchronization removes all the inter-node communication for state information exchange. It allows an efficient and scalable design for multiple-cloning without the need for explicit control, as we shall see in Section 3.1.3.

Figure 6 shows an example on how the synchronization is carried out. At the beginning (0), the original and the cloned socket have the same state (seq = nxt = 100). The state of the clone changes (1) when it sends out a message to the client (2). After the message is received by the client (3), it will send out an acknowledgement (4). The Packet Router, knowing the socket has been cloned, will look into the contents of the packet and update the socket's state accordingly (5). This step is done right above the device driver level and does not involve the operation of the normal network protocol stack.

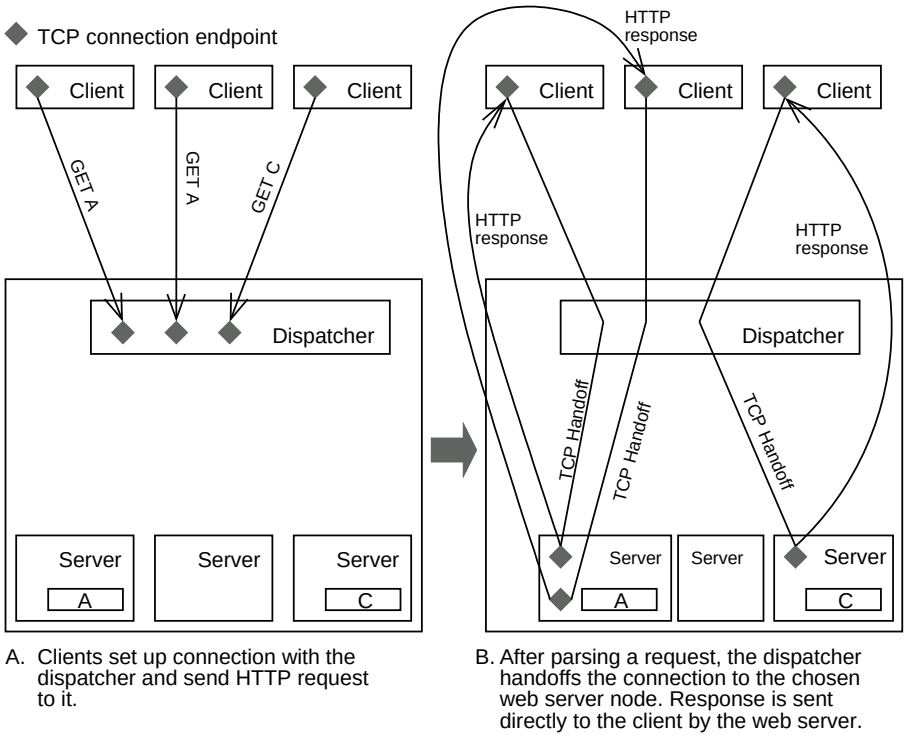

Figure 5: TCP Handoff

The acknowledgement is routed to the clone's node (6) by the Packet Router. This packet traverses the normal network stack in the clone's node and the states are updated (7). In this way, the states of the original and cloned socket are synchronized without explicit communications between the two cluster nodes. This reduces the overhead and improves the scalability of the system.

3.1.3 Persistent HTTP and Multiple Cloning

Handling non-persistent HTTP with SC is simple: the original node just decides whether to clone the socket by a mapping function. This function can be a hash function on the request, a load based function, a cache location mapping, or any other load balancing policies used in existing systems. It will either map the request to the node itself or another node in the cluster for processing. For persistent HTTP, an efficient and scalable mechanism is provided for cloning the socket multiple times so that every request is served by the node that has a cache of the document or the least load. Fine-grained load balancing can therefore be achieved.

The requests in a persistent HTTP connection are processed one after another in our system. When a clone handles a request, the web server in the clone's node will ignore the subsequent requests in that connection and close the clone after sending out the reply. The web server in the original node then handles the next request in that connection as if it is not in a persistent HTTP session: it can either clone the socket again, or it can handle the request itself. This process continues until all the requests within a connection have been handled. There is no limitation on the number of times that a socket is cloned, as long as there exists at most one clone of a particular socket at any time. In this way, multiple cloning is supported without the need to synchronize the web server nodes or to prevent the TCP stream from draining during cloning.

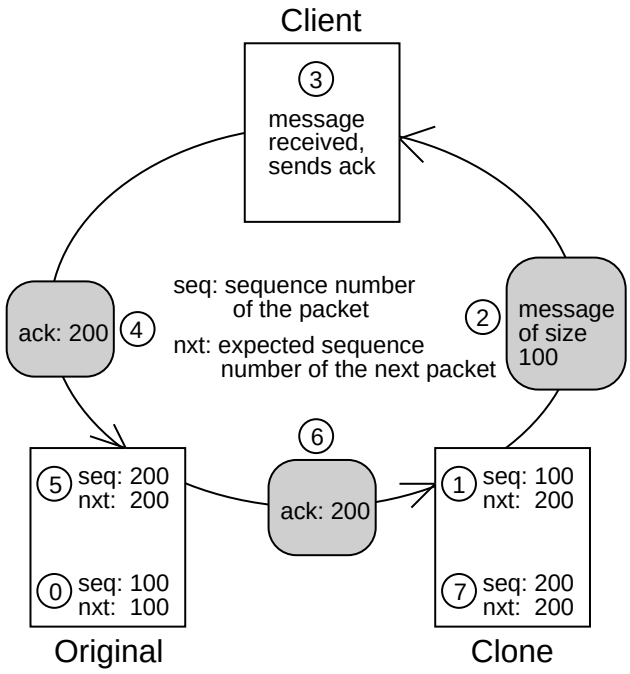

Figure 6: Triangular Implicit State Synchronization

Pipelining of request [35] is also supported in this approach by noting the length of the requested file by the cloning web server. This length is passed to the Packet Router by the cloning system call as an option. With this length, the message size (HTTP header and response) that the clone will send is computed. The Packet Router compares the acknowledged sequence number from the client and this pre-computed message size. The web server in the original node is informed by the Packet Router to process the next request when the whole reply has been acknowledged by the client. As the states of the clone and original sockets are synchronized automatically, the original node can take over the control of the connection right after the clone's node has finished serving the request. This ensures all the requests within a connection are served properly and efficiently without explicit control of the nodes.

Figure 7 summarizes the workflow of a cluster-based web server with Socket Cloning for a non-pipelined persistent HTTP connection.

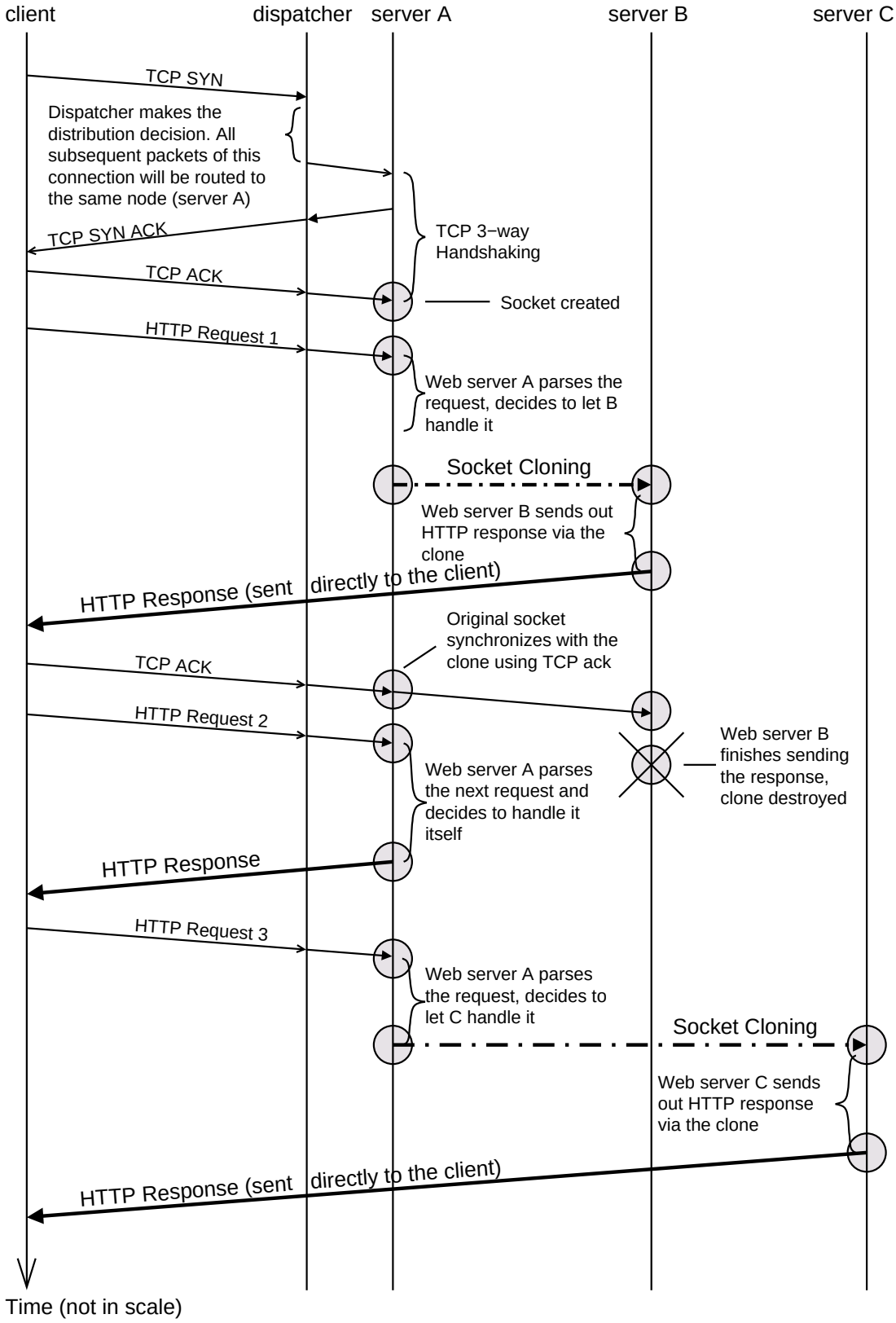

Figure 7: Workflow of a Cluster-Based Web Server with SC

3.1.4 Summary of Socket Cloning

There are several advantages of SC in cluster-based web servers:

1. The cloned socket can send packets directly to the client and outbound packets do not need to be forwarded to the original node first as in the relaying and Layer-4/3 approaches.
2. With SC, the processing of HTTP requests can be migrated to the node that has a cached copy of the requested document, thus bypassing any cache transfer between cluster nodes. As a result, the effective cache size is increased and network resource consumption is minimal.
3. Once cloned, the socket becomes native to the node and no extra network processing layer or wrapper is needed.
4. Multiple cloning of the same socket is allowed in Socket Cloning, which makes it possible to achieve fine-grained load balancing in persistent HTTP.
5. SC also has much lower overheads and thus higher scalability than the other approaches, as we shall see in later section.

Despite of the advantages of SC in cluster-based web servers, failure of server node will affect all the clones originated from it, as packets from the clients cannot be forwarded to the clones. This problem can be solved by monitoring the activity of the web server nodes at the dispatcher. When it receives no activity from a server node, it considers the node has failed and asks the remaining nodes to sent it the TCP ports and IP addresses of the clones originated from the dead node. The dispatcher then updates its routing table so that packets from the clients will be forwarded to the clone's node.

3.2 Hot Object Replication

In addition to Socket Cloning, Cyclone relies on the concept of hot object replication to achieve high performance by optimized processing of popular objects. Web server workloads are found to have a high concentration of references to a small group of documents [6, 7]. Only some of the documents in the web server are accessed frequently, while most of them are rarely requested. Analyses show that around 10% of the distinct documents are responsible for 80-95% of all requests received by the server in some workloads. Based on this observation, it is desirable to have those frequently requested documents, called hot objects, stored in the memory caches of some of the cluster nodes. The work needed to serve them is therefore shared. This provides better load balancing because some of the hot spots are removed. Throughput of the cluster is also improved, because the nodes can send the hot objects to the clients directly from it own cache.

3.2.1 Design of Hot Object Replication

Each web server node maintains a pre-allocated chunk of memory to store the replicated hot objects. This memory area is called hot object cache and is solely for storing replicated objects. Its size is relatively small to the system's memory because popular objects are concentrated in a very little set of documents. This has two advantages. First, the reduction of effective cluster memory size due to

replication is minimized. Second, the impact of preallocated memory on the system is also decreased as it leaves more space for both the kernel and user processes.

The general algorithm used in hot object replication is as follows. Each web server node maintains its reference counts of the set of documents that it is responsible for. This set includes the objects that are mapped by the function in Socket Cloning and the replicated objects that are stored in the node's memory. When the reference count of an object exceeds a predefined threshold, the document is considered 'hot' and will be replicated to other nodes by forwarding the cached copy. The forwarding mechanism is similar to cache forwarding in cooperative caching. When a web server node clones a socket that corresponds to a request for a hot object, the clone's node will send back the acknowledgement of cloning together with the cache copy of the document. It will then put the received copy in the hot object cache. Subsequent requests for that document are served from the node's cache. In this way, the document will eventually be replicated to all the nodes. Since all the nodes have a copy of the document, the reference count of it can be estimated in each node by incrementing with the number of replications, instead of one when it is requested. This eliminates the need to exchange messages for reference count updates of the replicated documents.

The cache replacement policy used in the hot object cache is LFU (least-frequently-used) with aging. All the reference counts of the documents are scaled down by an aging factor periodically. Documents that have built up high reference counts but rarely requested later can then be replaced more quickly. When a cache is evicted, all subsequent requests for that document will be handled by cloning the socket until it is added back to the cache again.

3.3 Prototype Implementation

We have implemented a prototype of Socket Cloning and hot object replication in Linux, kernel version 2.4.2. The network stack has been modified so that the clone (a socket) can be created without a real TCP three-way handshaking. This allows SC Server to perform a fake handshaking that does not involve packet exchanges to initialize the data structures of the network stack and the clone. A flag is also added to the socket's structure to differentiate a normal socket from a cloned socket. This is to avoid a clone to clone itself again. Normal network operations are not affected and applications are unaware of the change. SC Server, SC Client, and the Packet Router are all implemented as kernel modules. These modules have to be loaded in all the cluster nodes before any application can clone a socket. When the system starts, SC Client connects to the SC Servers in the other nodes of the cluster. All the SC Messages and cache copies are sent through these connections without the need to start a new one for each message. We have also modified kHTTPd [33], a kernel-based web server, to make use of the SC facilities.

Each web server node allocates a predefined size of memory to serve as the hot object cache. Currently, the size is 12 MBytes in a node with 128 MBytes of main memory. Such a small cache space reduces its impact on the cluster's effective memory size and it leaves more available memory to the system. The hot object cache is preallocated when the node boots up by an allocator module written by Alessandro Rubini [40] which allocates contiguous memory areas at the end of the physical memory. Arbitrarily large chunk of memory can be returned from this region to store the cache objects. These chunks are made available to the kernel space by ioremap(). Compared with the upper bound of 128 KBytes that get_free_pages() and kmalloc() can return, this approach provides better support for

large objects.

4 Micro-Benchmark of Socket Cloning

Socket Cloning plays an important role in achieving high performance in Cyclone. Its performance highly affects the cluster's throughput. To evaluate the performance of Socket Cloning, we have carried out tests with three content-aware mechanisms: SC, cooperative caching, and LARD [38]. Cooperative caching is implemented in kernel space in Linux for comparison because both LARD and SC run in kernel space. LARD is obtained from the project's homepage [8] and runs in FreeBSD 2.2.6.

4.1 Test Environment and Methodology

The tests were carried out in two two-node clusters: an Intel Pentium II 300MHz cluster and an Intel Pentium III 733MHz cluster. The cluster nodes and test client are connected via a Fast Ethernet switch. LARD is only tested in the 300MHz cluster due to some compatibility problems with the 733MHz cluster. In all the systems, a simple HTTP server is run.

In this test ApacheBench [4], a simple benchmark from Apache HTTP Server's source, was used. It measures performance by retrieving a single document repeatedly. The tested mechanism was applied to every request and the throughput of the whole system in terms of connections per second was measured. For example, in the test with cooperative caching, cache forwarding (CF) is always performed. Figure 8 illustrates how these tests were carried out.

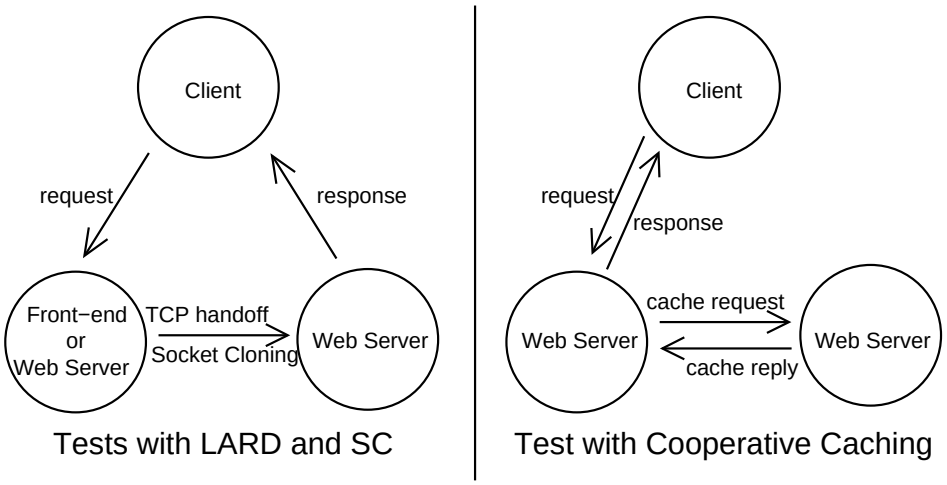

Figure 8: Micro-Benchmark Test

Since a single thread of ApacheBench was run in the tests, the reciprocal of the measured performance (connections/s) is the total time taken for a complete request-response pair. The overhead of a tested mechanism can thus be computed by subtracting this time from the time of a single web server running in the same environment.

4.2 Performance Analysis

It is found that Socket Cloning has the least overhead. The total overhead in sending the SC Message and setting up the clone is measured to be 134 μs in a Pentium II 300 node and 46 μs in a Pentium III 733 node, whereas it is 194 μs for Pentium II 300 machines in LARD. One may argue that these numbers depend on the operating system and do not represent the true cost. To minimize the effects of the different underlying environments and compare the relative overhead of the three mechanisms, a metric called *normalized efficiency* is used. To compute the normalized efficiency, throughput of the tested mechanism is first obtained. It is then divided by the throughput of a single web server running in the same environment. Figure 9 shows the result.

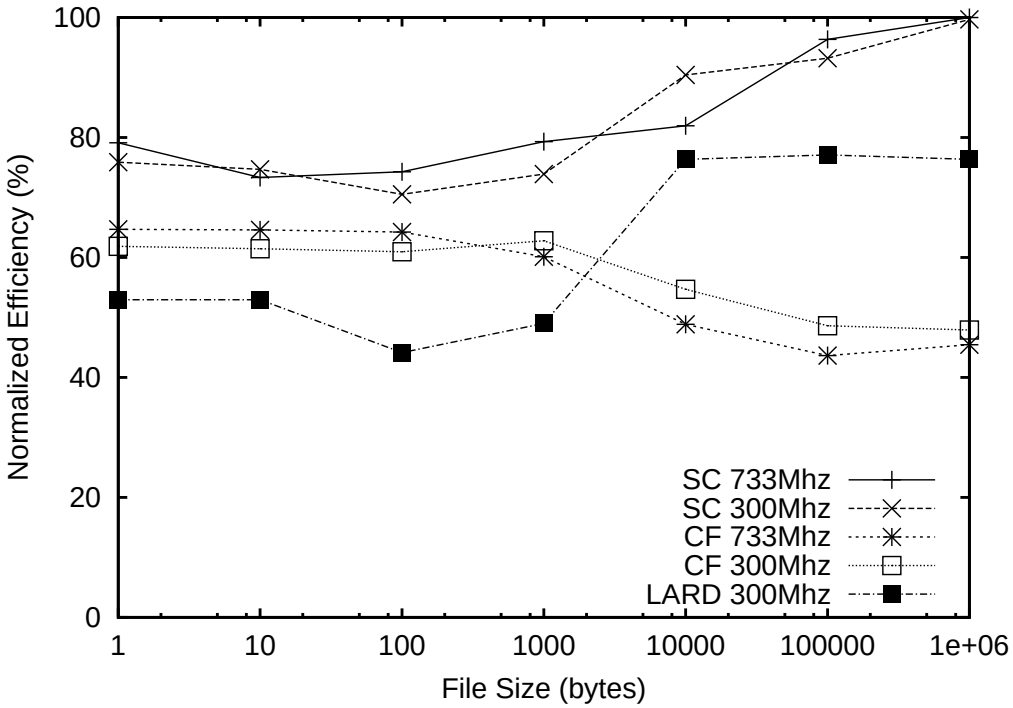

Figure 9: Normalized Efficiency

When the requested file is small, the startup cost of a mechanism is the main contributor of the decreased efficiency. The smaller normalized efficiency achieved in LARD shows that it have larger startup cost than the other two mechanisms.

In cooperative caching, the major cost is in cache forwarding. There are two messages involved in each CF: one is the request for a cache copy and the other is the cache reply. The normalized efficiency stays about the same for files of less than one kilobyte. It gradually drops to about 50% when larger files are transferred because the time to forward the cache becomes more significant. This shows the drawback of cache forwarding. The requested file is actually sent twice, once in cache forwarding and once in the reply. The crossover point of CF and LARD is about 3 kilobyte, which is similar to the result in [9].

In both of LARD and SC, there is a triangular routing path between the client and the request-handling web server node. Client packets have to be routed by an intermediate forwarding node in the cluster to reach the web server node while response packets are sent directly to the client. The

overhead in forwarding client acknowledgements thus becomes more significant in large response. In LARD, the IP address and TCP port number in the client packets have to be modified in both of the front-end and the handed off nodes. This adds extra delays in every client packet. As a result, it can only achieve less than 80% normalized efficiency. In SC, the Packet Router acts as a Layer-4/2 dispatcher and the clone's node does not have to modify the received packets. This scheme has lower overhead and SC is able to achieve nearly 100% normalized efficiency for large files.

Note that in real systems, SC or CF does not have to be carried out in all requests. If the node that the web client connects to has a cached copy of the document, then SC or CF is not necessary. Such situation can be increased if there are replications of cache copies in the cluster. On the other hand, the cost of TCP handoff has to apply to every connection in LARD because all the clients must establish connection with the front-end.

5 Trace-Driven Benchmark

We have also carried out trace-driven benchmark with cooperative caching (CF), SC alone, and Cyclone by replaying log files. Two cache replacement policies, LRU and LFU, were used in cooperative caching. CF with the hot object replication mechanism used in Cyclone (CF-REP) was also tested. It used LFU for non-replicated objects. LVS [46] with round-robin (RR) request distribution using direct routing, a Layer-4/2 mechanism, was tested as well for comparison. LVS was also used to distribute requests to the web server nodes in CF and SC. The mapping function used in SC is a hash function of the HTTP request. In all the tests, kHTTPd [33] ran in each of the server nodes.

5.1 Test Environment

Our testing environment consists of 1 to 8 web server nodes and 19 client nodes. All the machines are connected to a single Fast-Ethernet switch. Each of the servers has a 733MHz Intel Pentium III CPU and 128MB of main memory. Each client node runs http_load [30] with 2 concurrent connections. Http_load is modified to replay the log in order, rather than using the requests in the log randomly. All the files are stored in a 6-way SMP server connected in the same LAN and all the web servers mount the web documents by NFS.

The log file used is collected by the Computer Science Division, EECS Department, of UC Berkeley [47]. It recorded all the web accesses of the Division's website in January 2001. In our extracted log file, there are 74820 unique files with a total size of about 650 MB. The total number of requests is 3912120 and the average size of the requested documents is 7 KBytes.

Figure 10 shows the cumulative distributions of request frequency and size of the two log files. The distributions are similar to the general characteristics of web server workload [7, 6]. Hence, the result of our tests is representative for most of the real-life situations.

5.2 Benchmark Results and Performance Analysis

Figure 11 shows the results of the tests. The line Ideal in the figure is the total network bandwidth of the cluster. This is the limit of how much throughput one can get from the cluster and serves as an absolute reference point for the performance of the tested scheme.

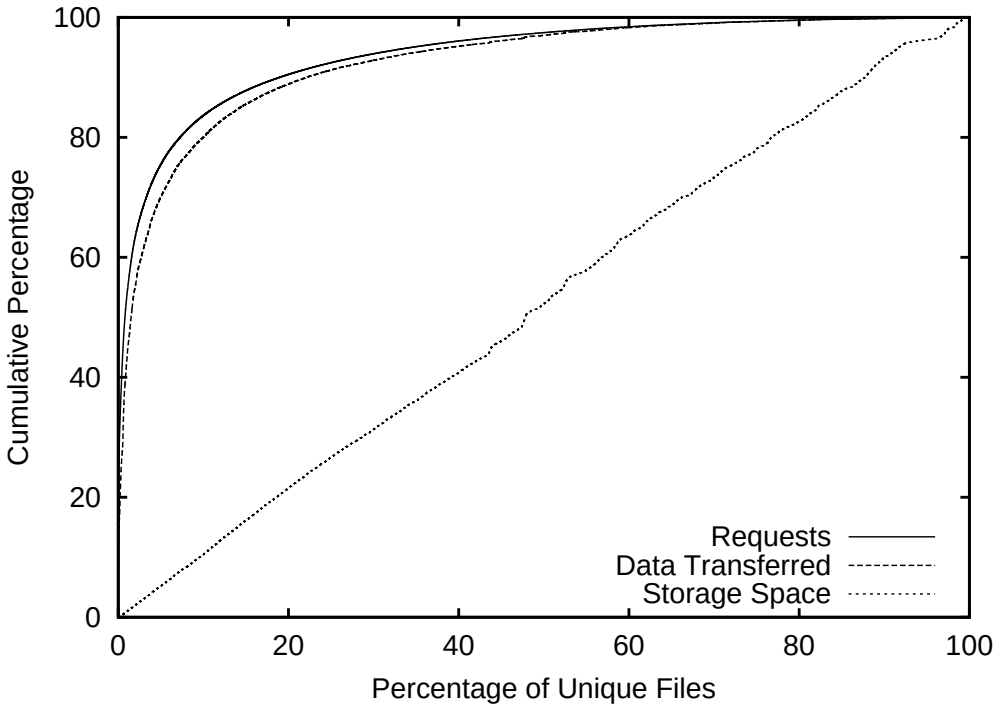

Figure 10: Concentration of References

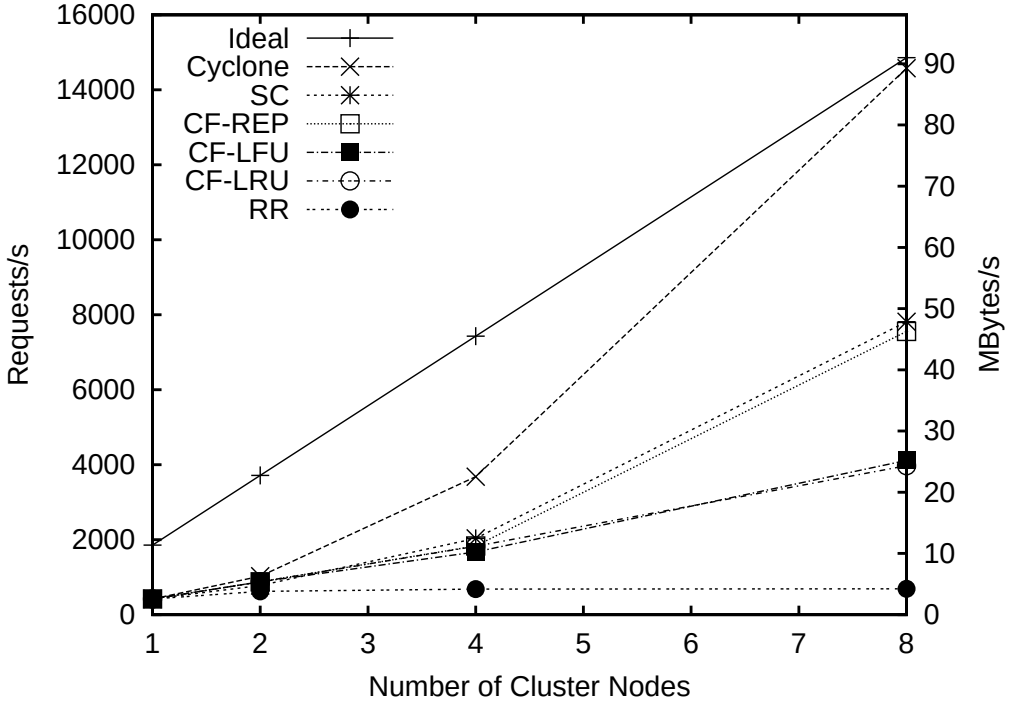

Figure 11: Throughputs in Trace-Driven Benchmark

5.2.1 Throughput

In the tests with less than eight nodes, the total size of the documents is larger than the sum of the memories in all the cluster nodes. The performance of a web server thus depends on how efficiently the memories of the nodes are used to avoid disk access. In these tests, throughput of RR does not scale with the number of web server nodes. The inefficient use of the memory in the cluster limits its scalability. Cooperative caching using LFU as the cache replacement policy has similar performance to the one that uses LRU. SC and CF-REP have similar performance with CF in the tests of two and four nodes. This is because of the small average document size. Retrieving a small file from the hard disk has much more overhead than transferring it to the client through the network. The total disk latency thus takes a large portion of the time of these tests and hence the benefits of SC and replication are hidden. The advantage of SC and CF-REP starts to show up with four nodes and SC outperforms CF by a large margin in the eight-node test.

In the eight-node test, there is no disk access in all the system except RR, because their total effective cache sizes are larger than the total document size. In this case, the difference in performance is completely due to the relative overhead of the schemes. In RR, a Layer-4 request distribution scheme, the effective cache size does not increase with the number of nodes in the cluster, which leads to very poor scalability of performance. Cooperative caching with LFU and LRU achieve similar performance because there is no replacement. CF-REP brings more performance improvement in this test than the LARGE test because of the smaller average file size. Cache forwarding is more efficient and the benefit of replication is thus more significant. However, CF-REP's performance is still inferior to SC and Cyclone. SC achieves 90% improvement over the performance of cooperative caching because of its lower overhead. With the efficiency of SC and hot object replication, Cyclone achieves nearly twice the throughput of SC alone and is able to obtain a throughput of 14575 requests/s (89.5 MBytes/s), which is about 98% of the available bandwidth of the cluster.

5.2.2 Efficiency

Figure 12 shows the efficiencies of these systems in the test. Efficiency is computed by dividing the system's throughput by the total available bandwidth of the cluster. This is an absolute indication of the cluster's performance. It represents the overall utilization of resources. With 98% efficiency, Cyclone accomplishes the goal of achieving high performance. Such performance cannot be achieved by CF and RR. Cache forwarding consumes too much network resources. In RR, memory is used inefficiently as described before. These drawbacks are reflected in the flat and decreasing efficiencies of CF and RR respectively. It is interesting to note that although the throughput of CF increases with the number of cluster nodes, its efficiency stays about the same. In CF-REP, SC, and Cyclone, efficiency keeps increasing.

5.2.3 Response Time

The average response times of the systems are shown in Figure 13. The response time is the time taken by the web server to return the first byte of the document after receiving the client's request. This measures the "latency" of the web server.

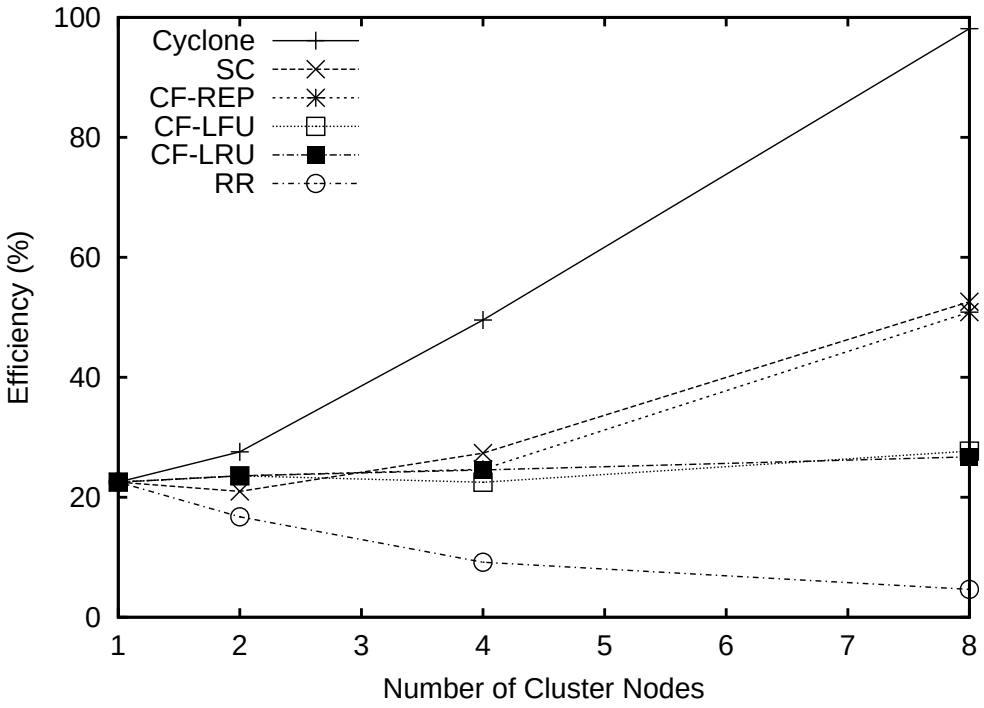

Figure 12: Efficiency

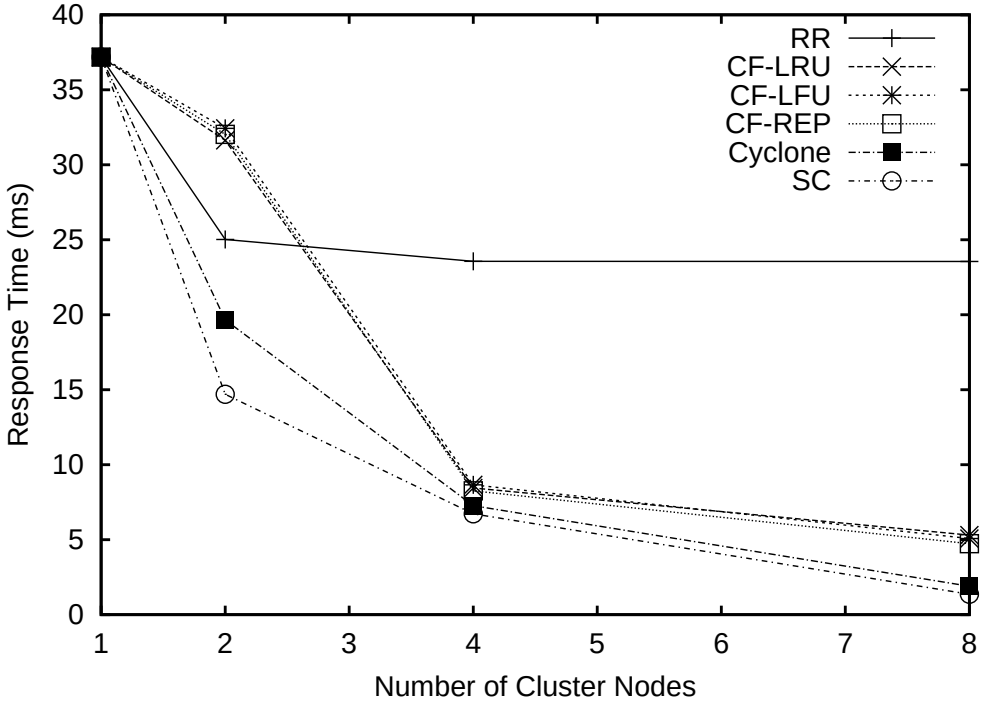

Figure 13: Response Times

In RR, there is little effect on reducing the response time with more than 2 nodes. This is largely due to the inefficient use of memory as discussed previously. All the CF systems have very long response time, especially when the cluster size is small. This is because of cache lookups and maintenance. The cache size of the cluster is much smaller than the total document size with a few nodes. Cache hit rate is low and therefore, searching for a cache copy in the peer nodes will usually fail. These ineffective lookup messages consume both network and computation resources and increase the response time of the system. When more nodes are added to the cluster and the cache hit rate increases, the response time shortens. CF-REP further reduces the number of lookup messages. It is still longer than SC and Cyclone, because of the extensive use of inefficient cache forwarding. The slow response of CF systems also explains partially why their performance lags behind those of SC and Cyclone.

Although Cyclone has much higher throughput than SC, it has longer response time. Each node in Cyclone has to maintain its hot object cache and forward caches to other cluster nodes. These are overheads to the system and they increase the response time by less than 1 ms in most cases. With the large performance gain, we believe such small overhead is justified.

6 Related Work

A lot of effort has been made in cluster-based web server to achieve high performance. However, previous systems only address one or two of the challenges that we have discussed.

In the Layer-4 dispatching approach [2, 22, 26, 29, 32, 42], request distribution is efficient. The dispatcher essentially acts as a router of web traffic by re writing packets. The overhead is small and is largely hidden by the latency of the Internet. Although this is a scalable approach, there is no cooperation among the web server nodes in the cluster. This leads to poor utilization of the cluster's memory. In addition, only coarse-grained load balancing can be achieved as request distribution is done at the TCP connection level. In Cyclone, Layer-4 approach can be used to distribute connections to the web server nodes, but the problems with it are solved by Socket Cloning. Cooperation of cluster nodes to efficiently utilize the cluster's memory is achieved by cloning a socket to the node that has a cache copy of the requested document. Fine-grained load balancing is realized by the support of multiple cloning. Replication of hot objects also helps to achieve high performance.

Layer-7 dispatching [17, 34, 38, 51] tries to solve the problem of ineffective memory utilization in its layer 4 counterpart. The dispatcher is able to parse the HTTP request before making the distribution decision. Requests of the same document are dispatched to the same node, which essentially partitions the web documents into several sets and increases the effective cache size. This requires the dispatcher to terminate the TCP connections with the clients and renders it to become the bottleneck of the cluster. Although there are several optimizations to Layer-7 dispatching [5, 19, 45, 49, 50], they only reduce the overhead in processing outbound packets. Dispatcher is still the only one in the cluster that actually sets up connections with clients. This limits the cluster's capacity. On the other hand, Cyclone does not depend on any particular dispatching methodology to achieve efficient memory utilization. Any request distribution scheme can be used to minimize the impact of the dispatcher.

The idea of cooperative caching [21] is applied to increase memory utilization in other systems [16, 20, 48]. The memories in the cluster nodes are treated as a single cache. When a web server node does not find a requested document in its own memory cache, it will ask other nodes for a cache copy.

If a node has it in cache, it will send the document to the web server node, which then forwards it to the client. While such approach can achieve effective memory utilization, it consumes too many network resources. In Cyclone, the socket is cloned to the node that has the cache copy. There is no actual cache transfer. As a result, Cyclone consumes much less network resources than systems with cooperative caching.

7 Conclusions

This paper has the following contributions:

1. The challenges that cluster-based web server has to overcome simultaneously to achieve high performance are identified.

2. A new cluster-based web server, Cyclone, which employs Socket Cloning and hot object replication to solve these problems is presented. This is the first system that addresses all of these problems at the same time. With eight nodes, the cluster can achieve a throughput of 14575 requests/s (89.5 MBytes/s) and the obtained efficiency is 98%, which is much higher than existing works.

3. An efficient and lightweight mechanism for Socket Cloning is proposed which incurs less overheads and system resources than existing approaches, such as TCP handoff and caching forwarding. This leads to a superior performance of our system.

4. We have also presented how multiple cloning is done to support fine-grained load balancing in persistent HTTP connections without synchronization and TCP stream drainage problems.

5. The performance metric on efficiency is introduced. This provides a new perspective on the real effects of up-sizing a cluster to the resultant performance.

We have discussed that a cluster-based web server has to meet the presented challenges simultaneously in order to achieve high performance. This is illustrated by the sub-optimal performance achieved when only SC is used. SC solves these problems except the processing of frequently accessed files. Although its performance is better than the cooperative caching approach by 90%, higher efficiency cannot be achieved even when all the documents are cached in the memory of the cluster. The sub-optimal performance achieved by cooperative caching with replication (CF-REP) also supports this argument. CF-REP attacks all the problems but consumes too much network resources. Leaving this unsolved leads to low throughput in the tests.

With these problems solved by SC and hot object replication in Cyclone, 98% of the total available bandwidth is achieved. This shows the importance of meeting all the challenges at the same time; leaving any of them unsolved will bring significant limitation to the resultant performance of the system. This has not been addressed in previous work and this leads to their inferior performance.

There are still some other optimizations left to be explored. For example, in persistent HTTP, the clone's web server can also parses the next request instead of closing the clone after serving a request. As a result, socket does not have to be cloned again if the next request is then determined to be handled by the clone's node. This will increase the load of the web servers, however, because both of

the original and clone socket parse the request. In addition, sockets can be pre-cloned in several nodes for processing the different requests within a pipelined persistent HTTP connection. Requests handling can thus be overlapped. When a web server has finished sending out the response, it can inform the clone in another node to serve the next request. The next response can thus be sent out immediately by that node without the latency in cloning the socket. While this can improve the performance, this will require more sophisticated protocols among the components in the Socket Cloning architecture.

Although this paper concentrates on cluster-based web server, we believe similar challenges still apply to other simple cluster-based network services such as ftp and web proxy. Architectures similar to Cyclone can provide a high performance solution to such service. In fact, Cyclone can easily be modified to become an efficient reverse proxy. we also believe that Socket Cloning can be beneficial to other computing paradigms, such as peer-to-peer networks. In peer-to-peer systems, forwarding of large messages host by host is a common way of delivering contents. With SC, forwarding is not necessary. A socket can be cloned to the communication endpoint that receives or sends the messages. Intermediate forwarders are no longer needed and the messages are exchanged in a direct point-to-point manner. This may require some modifications of the current design of SC and we are currently engaged with investigating such systems for possible solutions.

References

[1] W. H. Ahn, S. H. Park, and D. Park. Efficient cooperative caching for file systems in cluster-based web servers. *International Conference on Cluster Computing*, Nov. 2000.

[2] E. Anderson, D. Patterson, and E. Brewer. The Magicrouter: An appilcation of fast packet interposing. *Second Symposium on Operating Systems Design and Implementation*, May 1996.

[3] T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, Feb. 1995.

[4] Apachebench, http://www.cpan.org/modules/by-module/HTTPD/.

[5] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha. Design, implementation and performance of a content-based switch. In *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM-00)*, pages 1117–1126, Los Alamitos, Mar. 2000. IEEE.

[6] M. F. Arlitt and T. Jin. A workload characterization of the 1998 world cup web site. *IEEE Network*, 14(3):30–37, May/June 2000.

[7] M. F. Arlitt and C. L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, Oct. 1997.

[8] M. Aron. Scalable content-aware request distribution in cluster-based network servers, http://www.crpc.rice.edu/softlib/scalableRD.html.

[9] M. Aron, P. Druschel, and W. Zwaenepoel. Efficient support for p-http in cluster-based web servers. In *Proceedings of the USENIX 1999 Annual Technical Conference*, jun 1999.

[10] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of the USENIX 2000 Annual Technical Conference*, June 2000.

[11] A. Barak, O. La'adan, and A. Shiloh. Scalable cluster computing with mosix for linux. In *Proceedings of the 5-th Annual Linux Expo*, pages 95–100, May 1999.

[12] A. Bestavros, M. Crovella, J. Liu, and D. Martin. Distributed packet rewriting and its application to scalable server architectures. Technical Report 1998-003, CS Department, Boston University, Feb. 1 1998.

[13] T. Brisco. DNS suport for load balancing. *RFC 1794*, Apr. 1995.

[14] H. Bryhni, E. Klovning, and O. Kure. A comparison of load balancing techniques for scalable web servers. *IEEE Network*, 14(4):58–64, July/Aug. 2000.

[15] V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on Web-server systems. *IEEE Internet Computing*, 3(3):28–39, May/June 1999.

[16] G. Chen, C. L. Wang, and F. C. Lau. Building a scalable web server with global object space support on heterogeneous clusters. *International Conference on Cluster Computing*, Oct. 2001.

[17] L. Cherkasova and M. Karlsson. Web server cluster design with workload-aware request distribution strategy with ward. In *Proceedings of the 3rd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, pages 212–221, June 2001.

[18] Cisco Systens, Inc. Css 11100 content services switch, http://www.cisco.com/.

[19] A. Cohen, S. Rangarajan, and H. Slye. On the performance of TCP splicing for URL-aware redirection. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS-99)*, pages 117–126, Berkeley, CA, Oct. 11–14 1999.

[20] F. M. Cuenca-Acuna and T. D. Nguyen. Cooperative caching middleware for cluster-based servers. In *10th IEEE International Symposium on High Performance Distributed Computing*, Aug. 2001.

[21] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 267–280, Monterey, California, Nov. 1994.

[22] O. P. Damani, P. E. Chung, Y. Huang, C. Kintala, and Y.-M. Wang. ONE-IP: Techniques for hosting a service on a cluster of machines. *Computer Networks and ISDN Systems*, 29(8–13):1019–1027, Sept. 1997.

[23] F5 Networks, Inc. Big-IP Controller, http://www.f5.com/.

[24] R. Fielding, J. Gettys, J. Mogul, H. Nielsen, and T. Berners-Lee. Hypertext transfer protocol - HTTP/1.1. *RFC 2068*, Jan. 1997.

[25] Foundry Networks, Inc. ServerIron, http://www.foundrynetworks.com/products/ web-switches/serveriron/datasheets.html.

[26] X. Gan and B. Ramamurthy. LSMAC: An improved load sharing network service dispatcher. *World Wide Web*, 3(1):53–59, 2000.

[27] X. Gan, T. Schroeder, S. Goddard, and B. Ramamurthy. LSMAC and LSNAT: Two approaches for cluster-based scalable web servers. *IEEE International Conference on Communications*, pages 1164–1168, June 2000.

[28] X. Gan, T. Schroeder, S. Goddard, and B. Ramamurthy. LSMAC vs. LSNAT: Scalable cluster-based web servers. *Cluster Computing*, 3(3):175–185, 2000.

[29] S. Goddard and T. Schroeder. The SASHA architecture for network-clustered web servers. *Sixth IEEE International Symposium on High Assurance Systems Engineering*, pages 163–172, 2001.

[30] http_load. http://www.acme.com/software/http_load/.

[31] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. WireGL: A scalable graphics system for clusters. In *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 129–140. ACM Press / ACM SIGGRAPH, 2001.

[32] G. D. H. Hunt, G. S. Goldszmidt, R. P. King, and R. Mukherjee. Network Dispatcher: A connection router for scalable Internet services. In *Proceedings of the 7th International World Wide Web Conference*, Apr. 1998.

[33] kHTTPd - Linux HTTP accelerator. http://www.fenrus.demon.nl/.

[34] E. Levy, A. Iyengar, J. Song, and D. M. Dias. Design and performance of a web server accelerator. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-99)*, pages 135–143, Mar. 1999.

[35] J. C. Mogul. The case for persistent-connection HTTP. In *Proceedings of the SIGCOMM'95 conference*, Cambridge, MA, Aug. 1995.

[36] D. Mosedale, W. Foss, and R. McCool. Lessons learned administering Netscape's Internet site. *IEEE Internet Computing*, 1(2):28–35, Mar./Apr. 1997.

[37] Nortel Networks Ltd. Alteon ACEdirector, http://www.nortelnetworks.com/.

[38] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. *ACM SIGPLAN Notices*, 33(11):205–216, Nov. 1998.

[39] Resonate, Inc. Central Dispatch, http://www.nortelnetworks.com/.

[40] A. Rubini and J. Corbet. Linux device drivers, 2nd edition, http://examples.oreilly.com/linuxdrive2/ldd2-samples-1.0.1.tar.gz.

[41] Y.-F. Sit, C.-L. Wang, and F. Lau. Socket cloning for clustered-based web servers. *International Conference on Cluster Computing*, Sept. 2002.

[42] P. Srisuresh and D. Gan. Load sharing using network address translation. *RFC 2391*, Aug. 1998.

[43] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF : A parallel workstation for scientific computation. In *Proceedings of the International Conference on Parallel Processing*, pages 11–14, Boca Raton, USA, Aug. 1995. CRC Press.

[44] W. Tang, L. Cherkasova, L. Russell, and M. W. Mutka. Modular TCP handoff design in STREAMS-based TCP/IP implementation. In *Proceedings fo the First International Conference on Networking*, pages 71–81, July 2001.

[45] TCP Splicing. http://www.linuxvirtualserver.org/software/tcpsp/index.html.

[46] The Linux Virtual Server Project. http://www.linuxvirtualserver.org/.

[47] University of California, Berkeley. CS Access Log Files, http://www.cs.berkeley.edu/logs/.

[48] Whizz Technology Ltd. EC-Cache Engine, http://www.whizztech.com/ec_cache.html.

[49] C. Yang and M. Luo. An effective mechanism for supporting content-based routing in scalable web server clusters. In *Proceedings of the International Conference on Parallel Processing*, pages 240–245, Los Alamitos, CA, Sept. 1999. IEEE Computer Society.

[50] C. Yang and M. Luo. Building an adaptable, fault tolerant, and highly manageable web server on clusters of non dedicated workstations. In *Proceedings of International Conference on Parallel Processing*, pages 413–420, Washington - Brussels - Tokyo, Aug. 2000. IEEE.

[51] X. Zhang, M. Barrientos, J. B. Chen, and M. Seltzer. HACC: An architecture for cluster-based web servers. In *Proceedings of the 3rd USENIX Windows NT Symposium*, pages 155–164, Berkeley, CA, July 1999. USENIX Association.