

PRESS: A Clustered Server Based on User-Level Communication

Enrique V. Carrera, *Member, IEEE*, and Ricardo Bianchini, *Member, IEEE*

Abstract—In this paper, we propose and evaluate a cluster-based network server called PRESS. The server relies on locality-conscious request distribution and a standard for user-level communication to achieve high performance and portability. We evaluate PRESS by first isolating the performance benefits of three key features of user-level communication: low processor overhead, remote memory accesses, and zero-copy transfers. Next, we compare PRESS to servers that involve less intercluster communication, but are not as easily portable. Our results for an 8-node server cluster and five WWW traces demonstrate that user-level communication can improve performance by as much as 52 percent compared to a kernel-level protocol. Low processor overhead, remote memory writes, and zero-copy all make nontrivial contributions toward this overall gain. Our results also show that portability in PRESS causes no throughput degradation when we exploit user-level communication extensively.

Index Terms—Servers, communication architecture, performance.

1 INTRODUCTION

THE number of Internet users has increased rapidly over the last few years. This large user base is placing significant stress on the computing resources of popular services available on the Internet. Clusters of commodity computers are currently being used as the hardware infrastructure of most of these services. Thus, several researchers and companies have concerned themselves with cluster-based servers, e.g., [2], [3], [6], [11], [24], [25], [29], [39].

There are two main classes of servers in terms of how the clients' requests are distributed across the cluster: content-oblivious and content-aware servers. In a content-oblivious server, the request distribution is based solely on a load metric, which is usually the number of open connections being handled by each node. Early servers were mostly of this type. In contrast, a content-aware server makes request distribution decisions based on the actual content requested. Content-awareness allows for more informed and/or efficient resource management, such as in service differentiation (e.g., requests from certain clients are given high priority) or node specialization (e.g., certain cluster nodes serve images and others serve HTML files). Given the sophistication that content-awareness enables, these servers are becoming ever more popular.

In a content-aware server, the node that initially accepts a request (establishing a TCP connection with the client) and determines the content being requested may not be the node that should actually service the request. In that case, the request has to be forwarded to the most appropriate node, generating intracluster communication. The reply to

the request can either 1) be sent directly from the service node to the client or 2) be staged at the node that originally received the request. The former approach is efficient in that the potentially large reply does not have to be transferred between two cluster nodes, but requires a TCP connection hand-off mechanism [25] to transfer the TCP state of the node that accepts a request to the node that actually services the request transparently to the client. Unfortunately, operating systems currently do not provide TCP hand-off mechanisms, so implementing hand-off requires kernel modifications (as in [25]) or the loading of additional kernel modules to a STREAMS-based implementation of TCP (as in [32]). Hereafter, we refer to servers that rely on TCP hand-off as *hand-off servers*.

The latter approach is to transfer each reply to the node that originally received the corresponding request, which can, in turn, relay the reply to the client. This approach does not require any TCP hand-off infrastructure, but is potentially less efficient. The extent of the performance degradation caused by the extra data traffic depends mainly on the performance and characteristics of the internode communication subsystem. Hereafter, we refer to servers that do not rely on TCP hand-off as *request-reply servers*.

Content-aware distribution can be exploited to improve cache locality, as originally proposed in [25]. The idea is to aggregate the set of memories of the cluster into a large cache and distribute requests for files based on cache locality, as well as load balancing considerations. For their focus on cache locality, we refer to these servers as *locality-conscious servers*.

In this paper, we propose and evaluate PRESS [12], [13], a portable and efficient locality-conscious, request-reply WWW server. To achieve high performance, PRESS promotes both cache locality and load balancing without any centralized resources, relying heavily on user-level communication for intracluster messaging. To guarantee portability, PRESS is implemented completely in user space and uses the Virtual Interface Architecture (VIA) user-level

- E.V. Carrera is with the Colegio Politecnico, Universidad San Francisco de Quito, PO Box 17-12-841, Quito, Ecuador. E-mail: vimicioc@usfq.edu.ec.
- R. Bianchini is with the Department of Computer Science, Rutgers University, 110 Frelinghuysen Rd., Piscataway, NJ 08854-8019. E-mail: ricardob@cs.rutgers.edu.

Manuscript received 11 Sept. 2003; revised 11 Aug. 2004; accepted 14 Oct. 2004; published online 20 May 2005.

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0162-0903.

standard for intracluster communication. VIA has been implemented for a wide variety of proprietary system-area networks, such as Myrinet, ServerNet, and Giganet, as well as for Fast and Gigabit Ethernet.

We evaluate PRESS by first assessing the impact of the characteristics of the communication subsystem on the performance of the server. In particular, we isolate the throughput benefits of three key features of user-level communication that are available in VIA: low processor overhead, remote memory accesses, and zero-copy transfers. Our results for an 8-node cluster and five WWW traces demonstrate that user-level communication can increase throughput by as much as 52 percent (43 percent on average), compared to a kernel-level protocol such as TCP. Low processor overhead, remote memory writes, and zero-copy all make nontrivial contributions toward this overall gain. Next, we compare PRESS to two locality-conscious servers based on TCP hand-off and a traditional collection of independent (locality-oblivious) servers. Our experimental results show that PRESS performs as well as the servers based on hand-off when it exploits remote memory accesses and zero-copy transfers for all its messages. The three locality-conscious servers easily outperform the cluster of independent servers.

Based upon our experience and results, we conclude that features such as remote memory accesses and zero-copy transfers provide significant performance gains for content-aware servers. These features should continue to be supported by interconnection infrastructure vendors. Furthermore, we conclude that implementing servers completely in user space need not entail any performance degradation; kernel-level hand-off infrastructure is not really required, if efficient user-level communication is available. Thus, service providers need not run the risk of having to reimplement their servers for each new (generation of) operating system or hardware platform.

In summary, we make the following contributions:

- We describe the design and implementation of PRESS, a locality-conscious network server that exploits user-level communication. As far as we know, PRESS is the first clustered server based on user-level communication.
- Through a detailed evaluation of PRESS, we demonstrate that user-level communication can be used to promote server portability and high performance at the same time. Previous works strived to achieve high performance at the cost of complex operating system extensions.
- In our evaluation, we also quantify the performance benefits of each feature of user-level communication for a sophisticated server. Previous works on user-level communication have only considered micro-benchmarks and parallel scientific applications.

The remainder of the paper is organized as follows: The next section presents a little background on user-level communication, VIA, and locality-conscious servers. Section 3 presents PRESS in detail. Section 4 presents the methodology used in our experiments and discusses our performance results. Section 5 discusses the related work.

Finally, Section 6 summarizes our findings and concludes the paper.

2 BACKGROUND

2.1 User-Level Communication and VIA

The goal of user-level communication is to reduce the software overhead involved in sending and receiving messages by removing the operating system from the critical communication path in a protected fashion. This goal is achieved by giving processes direct access to their network interfaces; the operating system is only involved in setting up the communication. In this way, the communicating parties can avoid intermediate copies of data, interrupts, and context switches in the critical path. The result of these optimizations is communication latency and bandwidth that approach the hardware limits.

There has been extensive research in user-level communication, e.g., [1], [7], [8], [15], [27], [33], [35], [36]. This research led to the proposal of the VIA [14], [16] industry standard by a group of leading computer companies. In VIA, each communicating process can open directly accessible interfaces to the network hardware. Each interface, called *Virtual Interface* (VI), represents a communication end-point analogous to the socket end-point in a traditional TCP connection. In this way, pairs of VIs can be connected to create communication channels for bidirectional point-to-point data transfers. Each VI has a send and a receive queue. Processes post requests to these queues to send or receive data. The requests have the form of descriptors. Each descriptor contains all the information that the network interface controller needs to process the corresponding request, including pointers to data buffers. The network interface controller asynchronously processes the posted descriptors and marks them when completed. The processes then remove completed descriptors from the queues and reuse them for subsequent requests. In order to facilitate the removal of completed descriptors, VIA also specifies *Completion Queues* (CQs). A CQ can combine the notification of descriptor completions of multiple VIs into a single queue.

VIA provides two styles of communication. Besides the traditional send/receive data transfer model, it allows for direct access to the memory of remote machines. A remote memory access reads/writes data from/to the correct remote addresses without intervention by the remote processor. Regardless of the style of communication, the memory used for every data transfer in VIA needs to be “registered.” The registration locks the appropriate pages in physical memory, allowing for direct DMA operations in the user memory buffers without the possibility of an intervening page replacement.

Finally, the VIA standard specifies three levels of reliability: unreliable delivery, reliable delivery, and reliable reception. In unreliable delivery, both regular and remote memory messages can be lost without being detected or retransmitted. In reliable delivery, all data submitted to transfer are guaranteed to arrive at the destination network interface exactly once and in order, in the absence of errors. If an error occurs, it is reported. Reliable reception is similar

to reliable delivery, but a transfer is only successfully completed when the data have been delivered to the target memory location.

VIA has several successful implementations for both custom and Ethernet adapters, e.g., Myrinet-VI [10], Giganet VIA [17], M-VIA [9], ServerNet VIA [30], and FirmVIA [4]. Our experiments use Giganet VIA. This implementation supports VIA in hardware using its proprietary cLAN network interfaces connected through 2.5 Gbits/s full-duplex links. Giganet VIA does not support remote memory reads (only remote writes are supported) and reliable reception.

Recently, InfiniBand [34] has appeared as the new standard for I/O connectivity. InfiniBand is a switch-based point-to-point interconnect architecture. Among other important features, InfiniBand supports high-speed, low-latency communication based on VIA-like execution queues and remote memory accesses (reads and writes). The standard also defines that nodes can communicate at 2.5, 10, or 30 Gbits/s. Although we experiment with VIA, our study should apply to InfiniBand as well.

2.2 Locality-Conscious Servers

Locality-conscious servers use the set of memories of the cluster as a single large cache and distribute requests for files based on cache locality. This strategy has the potential to improve performance significantly with respect to traditional, locality-oblivious servers since serving a request from main memory (even if the memory is remote) is much less expensive than serving it from a local disk. Locality-conscious servers are useful when the working set of services exceeds the size of local memories in the cluster. In fact, attempting to reduce cache miss rates by simply oversizing main memories (and constantly buying more memory) does not seem economical—or even viable beyond a certain point—for several services, such as WWW hosting and e-mail services, given how quickly their working sets can grow. Even though we focus on locality-conscious servers, our study should apply to any type of content-aware server.

3 PRESS

3.1 Overview

PRESS is our portable cluster-based, locality-conscious WWW server [12]. Although WWW servers must service requests for both static and dynamic contents, in this paper, we focus solely on PRESS as a server of static content (read-only files) since this type of content puts the most stress on the intracluster network.

Just like other locality-conscious servers [3], [6], [25], PRESS is based on the observation that serving a request from any memory cache, even a remote cache, is substantially more efficient than serving it from disk, even a local disk. Essentially, the server distributes HTTP requests across the cluster nodes based on cache locality and load balancing considerations so that the requested content is unlikely to be read from disk if there is a cached copy somewhere in the cluster.

The pseudocode for request distribution in PRESS is shown in Fig. 1. PRESS assumes that HTTP requests are

```

1. while (true)
2.   receive and parse next request  $r$ ;
3.   if ( $filesize(r.target) \geq max\_size$ ) then
4.     serve  $r$  but do not cache  $r.target$ ;
5.     continue;
6.   if ( $Servers[r.target] = \emptyset$ ) then
7.     add  $initial\_node$  to  $Servers[r.target]$ ;
8.      $service\_node \leftarrow initial\_node$ ;
9.   else
10.    if ( $initial\_node \in Servers[r.target]$ ) then
11.       $service\_node \leftarrow initial\_node$ ;
12.    else
13.       $service\_node \leftarrow \{least\ loaded\ node\ in\ Servers[r.target]\}$ ;
14.    if ( $service\_node.load > T$ ) then
15.      if ( $initial\_node.load < T$ ) then
16.        add  $initial\_node$  to  $Servers[r.target]$ ;
17.         $service\_node \leftarrow initial\_node$ ;
18.      else
19.         $idle\_node \leftarrow \{least\ loaded\ node\}$ ;
20.        if ( $idle\_node.load < T$ ) then
21.          add  $idle\_node$  to  $Servers[r.target]$ ;
22.           $service\_node \leftarrow idle\_node$ ;
23.    if ( $service\_node = initial\_node$ ) then serve  $r$ ;
24.    else forward  $r$  to  $service\_node$  (reply will come later);

```

Fig. 1. Pseudocode for request distribution in PRESS.

directed to the cluster using a standard method, such as Round-Robin DNS or a content-oblivious, load balancing front-end device. Thus, any node of the cluster can receive a client request and become the *initial node* for that request. When the request arrives at the initial node, the request is parsed (line 2 in Fig. 1) and, based on its content, the node must decide whether to service the request itself or forward the request to another node. A request for a large file (size ≥ 256 KBytes in our prototype) is always serviced locally by the initial node (lines 3-5). In addition, the initial node is chosen as the *service node* if this is the first time the file is requested (lines 6-8) or it already cached the requested file (lines 10-11). If neither of these conditions hold, the least loaded node that is caching the requested content becomes a candidate for the *service node* (line 13). This node is chosen as the *service node* either when it is not overloaded (i.e., its number of open connections is not larger than a user-defined threshold, $T = 256$ in our experiments) or when it is overloaded but so are the initial node and the least loaded node in the cluster (lines 14-22). In this way, popular files end up replicated at multiple nodes for better load balancing.

The overhead of this request distribution algorithm is negligible compared to the cost of receiving and processing each request, even for large clusters. The only operation that cannot be performed in constant time is line 13 of Fig. 1. Nevertheless, the vast majority of files is typically cached by a single node, making this operation very efficient in practice.

A forwarded request is handled in a straightforward way by the *service node*. If the requested file is cached, the *service node* simply transfers it to the *initial node*. If the file is not cached, the *service node* reads the file from its local disk, caches it locally (becoming part of the server set for the file), and, finally, transfers it to the *initial node*. Upon receiving the file from the *service node*, the *initial node*

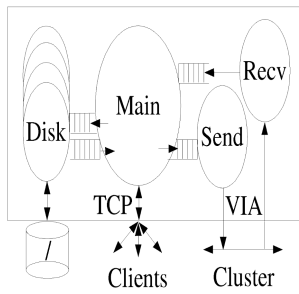


Fig. 2. PRESS communication architecture.

sends it to the client. The initial node does not cache the file received from a service node to avoid excessive file replication across the cluster.

In order to be able to intelligently distribute the HTTP requests it receives, each node needs locality and load information about all the other nodes. Locality information takes the form of the names of the files that are brought into the caches, whereas load information is represented by the number of open connections handled by the nodes.

The dissemination of caching information is straightforward. Whenever a node either replaces or starts caching a file, it broadcasts that fact to the other nodes. Broadcasts of caching information are very infrequent in steady-state.

The dissemination of load information can be done in one of two ways: broadcasting and piggy-backing. When broadcasting, each node broadcasts its current load when the load is a certain number of connections (the load threshold) greater or smaller than the last broadcast value. When piggy-backing, the current load is appended to any message sent among the nodes. In this way, each message updates the load information of its sender on the receiver. We have shown in [13] that piggy-backing performs at least as well as broadcasting, so, in this paper, we use piggy-backing.

PRESS compares favorably against other servers in terms of performance. Its single-node throughput is equivalent to that of the Flash server [26] as these two servers are based on similar optimizations. Flash has been shown superior to Apache in [26]. In addition, note that PRESS can be used without modification to achieve a good distribution of requests for either persistent (such as HTTP/1.1 [18]) or nonpersistent (such as HTTP/1.0 [5]) connection-oriented protocols. Section 4 compares PRESS to other servers for clusters.

3.2 Communication Architecture

Fig. 2 illustrates the PRESS architecture. The local server thread at each node is event-driven. For highest performance, this main thread should never block, so we use helper threads for operations that can potentially block (disk access and intracluster communication). Client/server communication is implemented with nonblocking socket sends and receives in the main thread. A Unix `select()` statement in the main thread makes sure that the server does not block for socket communication. The file data are mapped to memory to avoid double buffering.

In order to communicate inside the cluster, each cluster node sets up VI end-points with each other node. Two threads per node are responsible for sending and

receiving messages; we refer to these threads as the send thread and the receive thread, respectively. These threads remain blocked until there is a message to be sent or received. The main server thread unblocks the send thread, using a semaphore, after a *digest* of the message to be sent has been queued at a data structure that is shared by these two threads. When the send thread wakes up, it simply creates a message descriptor and places it at the corresponding send queue. The receive thread is unblocked by the arrival of any regular (as opposed to remote memory write) message at one of the receive queues. When the receive thread wakes up, it simply determines which VI received the message and places a digest of the message at a data structure shared with the main thread. The main thread periodically polls this data structure. Remote memory writes have no effect on the receive thread. In fact, the receive thread is not even required if all the communication is effected with remote memory writes.

Intracluster communication in PRESS occurs for six types of messages:

- exchange of load information—very short messages carrying numbers of open connections;
- exchange of caching information—short messages carrying file names;
- request forwarding—short messages carrying file names;
- file transfer—long messages carrying file data;
- window-based flow control—very short messages carrying numbers of empty buffer slots; and
- buffer management—very short messages carrying the address and size of used buffers.

For simplicity, we group flow control and buffer management messages into a single category (flow control messages). In addition, as we explained above, load information exchanges are not required when we piggy-back load information to other messages. Thus, we do not experiment with this type of message. In Section 4, we experiment with several possible implementations for the other message types. The implementations differ in terms of the extent to which they use remote memory writes and copying.

Implementations that use remote memory writes can improve performance by not involving the receiver in the transfer. However, remote memory writes can also require more messages if copies are to be avoided. Furthermore, remote memory writes can involve significant polling overhead, especially for cluster configurations with a large number of nodes when messages cannot be overwritten and must be handled quickly.

As an example of this trade-off, a request forwarding message has to be handled quickly to reduce latency, so the main thread has to poll several queues (one queue for each node) fairly frequently to determine whether another node needs to receive a file. The regular message implementation of this message type reduces the amount of polling to the main thread polling of the single data structure it shares with the receive thread. However, this latter implementation requires data copying from the message descriptor to the shared data structure so that the descriptor can be reutilized by another message as quickly as possible, reducing flow control stalls.

In contrast, no overhead is associated with remote memory writes for messages implementing flow control or carrying load information. These messages do not require immediate attention and can be overwritten, which makes their implementation with remote memory writes much more efficient than with regular messages.

Besides the VIA implementation, we also have a prototype of PRESS that uses TCP for comparison purposes. The TCP version basically has the same structure of its VIA counterpart; the main differences are the replacement of the VI end-points by TCP sockets and the elimination of flow control messages, which is implemented transparently to the server by TCP itself.

4 EXPERIMENTAL EVALUATION

We start the evaluation of PRESS by discussing our experimental setup and the systems used for comparison. After that, we assess the performance of PRESS as a function of the characteristics or features of the communication subsystem. In particular, we isolate the benefits of low processor overhead, remote memory accesses, and zero-copy transfers. We also compare PRESS to hand-off and traditional servers. We close the section by summarizing our results and discussing a few alternative evaluation scenarios.

4.1 Methodology

Cluster hardware and workloads. Our server cluster is comprised eight Linux-based PCs with 800-MHz Pentium III processors, 512 KBytes of memory, an SCSI disk, and an interface to a Gigaset (cLAN) switch. Besides our server cluster, we use another eight Pentium-based machines (called clients) to generate load for the servers; eight servers and eight clients is the largest configuration that our switch can support with full bandwidth. These clients communicate with the server using TCP over the cLAN switch. This arrangement is essentially equivalent to using separate switches for external and intracluster communication since the cLAN is not a performance factor (its utilization is always less than 25 percent) in our experiments. For simplicity, we did not experiment with a content-oblivious front-end device; we used the equivalent strategy of having clients send requests to the nodes of the server in a randomized fashion with equal probabilities.

The clients reproduce five WWW server traces with requests from thousands of real users. All these traces have been used in previous studies. Clarknet is a trace from a commercial Internet provider, Nasa is from NASA's Kennedy Space Center, Rutgers contains the accesses made to the main server for the Computer Science Department at Rutgers University in the first four months of 2002, Usask contains seven months of requests to the University of Saskatchewan, and WC98 corresponds to three days of accesses to the 1998 Soccer World Cup WWW server. To avoid overestimating the benefits of user-level communication, we assume all requests in the traces require a new TCP connection. This assumption increases the percentage of time that processors spend in connection establishment and termination with the clients, which user-level communication does not address.

TABLE 1
Main Characteristics of the WWW Server Traces

Logs	Num files	Avg file size	Num requests	Avg req size	α
Clarknet	34126	12.0 KB	3327950	9.3 KB	0.79
Nasa	9129	28.1 KB	3461567	22.1 KB	0.94
Rutgers	70256	23.0 KB	4765224	19.4 KB	0.89
Usask	13760	14.4 KB	2408174	6.2 KB	0.61
WC98	24328	9.1 KB	51117263	4.2 KB	0.77

We eliminated all incomplete requests in the traces and ended up with the characteristics listed in Table 1. The traces cover a relatively wide spectrum of behavior. The number of files ranges from about 9K to 70K, with an average file size between roughly 9 and 28 KBytes. The average size of the files serviced ranges from about 4 KBytes to 22 KBytes. The coefficient of the Zipf-like distribution (α) that better represents each trace also varies widely, from 0.61 to 0.94. However, these traces have very small working sets compared to those of today's popular WWW content providers or WWW hosting services. Thus, we use small main memory caches (192 MBytes for Clarknet, 96 MBytes for Nasa, 352 MBytes for Rutgers, 48 MBytes for Usask, and 64 MBytes for WC98) in the experiments with these traces to reproduce situations where working set sizes are significant in comparison to cache sizes. The memory cache sizes were picked to produce miss rates in the 1-4 percent range when a traditional (locality-oblivious) server is run on eight nodes.

Our measurements are taken after a short cache warm-up run. To determine the maximum throughput of each system, our measurements disregard the timing information in the traces, making clients issue new requests as soon as possible. Under these assumptions, the nodes' CPUs are the throughput bottleneck for the locality-conscious servers, regardless of the cluster size.

Note that we focus on throughput (and messaging behavior) since server latencies are typically low compared to the overall delay a client experiences establishing connections, issuing requests, and waiting for replies across a wide-area network. Nevertheless, we do discuss latencies in Section 4.4.

Locality-oblivious and hand-off servers. We compare PRESS to a traditional locality-oblivious server and two hand-off servers. The locality-oblivious server uses the same code as PRESS, but does not run the request distribution algorithm and does not involve any information dissemination between nodes as nodes service the requests they receive completely independently of other nodes.

The two hand-off servers are organized differently. One of them is similar to PRESS in that request distribution decisions are made by each node independently. If the initial node decides that the request should be serviced by another node, it hands off the connection to the other node. As distribution decisions are distributed, this server uses broadcasting for exchanging caching information and piggy-backing for load information, just as PRESS does.

In contrast, the other hand-off server uses a centralized dispatcher process (which runs on a ninth cluster node to avoid any interference with server nodes) that makes all distribution decisions, as proposed in [3]. In their study, any

node can receive a request from a client, but it has to communicate with the dispatcher to find out which node should actually service the request. After experimenting with this basic strategy, we found that performance is improved when communication with the dispatcher happens only on local cache misses. Thus, our dispatcher-based server uses this optimization. (Our server also attempts to coalesce messages to the dispatcher, but this feature is rarely exercised in our experiments.) If the dispatcher decides that a node other than the initial node should service the request, the initial node hands off the connection to the other node. As the distribution decisions are only made by the dispatcher, this server obviates the need to disseminate load and caching information to other nodes. The request distribution algorithm used by both hand-off servers is the same as that of PRESS to make comparisons possible. Intracluster communication in the hand-off servers is implemented with regular VIA messages over Giganet. Remote memory writes and zero-copy messages are effectively useless for these servers as they do not transfer actual file data inside the cluster.

To study TCP hand-off under the best possible light, our hand-off servers implement an overhead-free emulation of this mechanism. We emulate a TCP hand-off by transferring 16 bytes (besides the request info) over VIA/Giganet for each forward operation. No overhead is assumed to retrieve and set the TCP connection state. The 16 bytes represent a minimal TCP state. The service node replies directly to the requesting client over permanent TCP connections. All the code of the hand-off servers, besides the forwarding operation, is the same as that of PRESS.

4.2 Performance as a Function of Communication Characteristics

Effect of processor overhead. In order to understand the effect of processor overhead, we compare the performance of PRESS running on TCP and VIA over the same Gigabit interconnect, the cLAN switch. Regardless of the protocol used for intracluster communication, the communication with the clients is done through TCP over the cLAN switch. Recall that the cLAN bandwidth is high enough that it can easily accommodate both intracluster and external communication without noticeable contention.

Fig. 3 plots the throughput of PRESS for our five traces under the two different protocols on eight nodes. We labeled these two versions of PRESS "TCP" and "Base-VIA." In more detail, these two implementations have the following characteristics:

- TCP. The server uses TCP over cLAN for all messages. The TCP over cLAN implementation runs the complete TCP stack and does not take advantage of the reliability and message ordering properties of the cLAN. In this implementation, sending a 4-byte message takes 76 microseconds and the observed communication bandwidth is 32 MBytes/s with 32-KByte messages.
- Base-VIA. The server uses VIA over cLAN for all the intracluster communication. In the basic implementation depicted in the figure, neither remote memory writes nor zero-copy transfers are used.

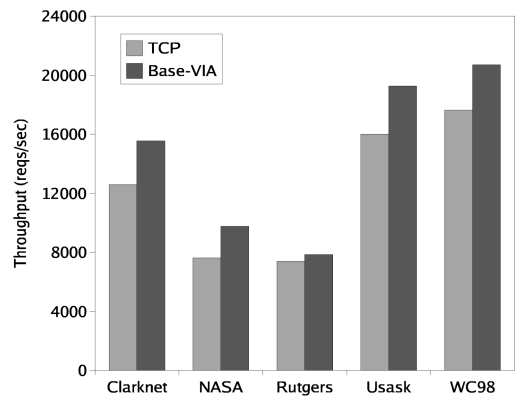


Fig. 3. Effect of processor overhead in TCP and VIA on throughput.

Sending a 4-byte message takes 16 microseconds and the network peaks at 102 MBytes/s bandwidth for 32-KByte messages. (When using remote memory writes and polling, a 4-byte message takes only eight microseconds.)

By comparing TCP and Base-VIA, we isolate the effect of processor overhead on performance since the VIA overhead is a factor of almost five lower than that of TCP. This comparison shows that the gains that can be accrued by exploiting a protocol with lower processor overhead can be substantial, ranging from 6 percent for Rutgers to 28 percent for Nasa.

Effect of remote writes and zero-copy. To evaluate the usefulness of remote memory writes and zero-copy transfers, we developed three other versions of PRESS: "RDMA-1c," "RDMA," and "Zero." The version we studied so far, labeled "Base-VIA" in Fig. 3, is our base version ("Base"). The versions differ in the extent to which remote writes and zero-copy are utilized. In all versions, load information is piggy-backed in other messages and intracluster communication uses VIA over cLAN.

Version Base only utilizes regular messages, i.e., a receiver is interrupted to handle an arriving message and data copies are made at the sender and at the receiver for all message types. Note that most of the data copies in version Base are of no serious performance consequence (very little data are involved in each case), except for the copies made at the sender and receiver of a file transfer message. The copy at the receiver side is unavoidable when using regular messages since files can be bigger than message descriptors and the corresponding descriptor must be freed as quickly as possible for use by another message. The copy at the sender side could be avoided, provided that we could either make all cache pages unswappable and available to VIA or manage a smaller cache of unswappable and available pages with low overhead.

Version RDMA-1c improves on version Base by using remote memory writes instead of regular messages. This version does not require a receive thread as all communication is performed with remote memory writes. In flow control messages, only a word of data is transferred per message. The messages do not require immediate attention and can be overwritten. Forward and caching messages are very similar in that a file name is transferred in each

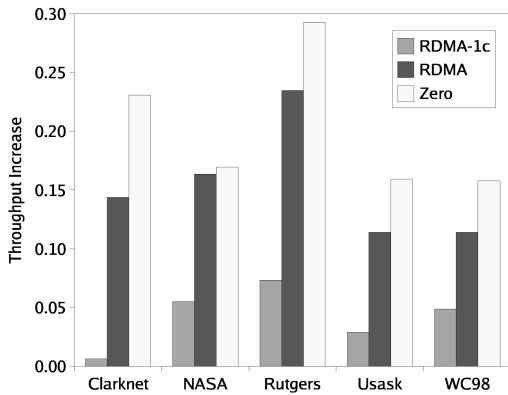


Fig. 4. Effect of VIA RMW and zero-copy. Throughput increase with respect to Base.

message, the messages cannot be overwritten, and polling fairly frequently for these messages can improve performance. At each node, two circular buffers are allocated for forward and caching messages for each other node. As each node knows the location of its private buffers at every other node, it keeps track of exactly where the next file name should be written in the memories of remote nodes. Polling is done by looking at message sequence numbers stored at the last position of each (fixed-size) buffer entry. A receiver determines that the next message has not yet been completely received if its sequence number is not equal to the last sequence number plus 1. Processors poll for these messages at the end of the main server loop. The data structures for file transfers are a small circular buffer that is similar to the forward and caching buffers just described and a large circular buffer for the actual file data. These two buffers are allocated by each node for each other node. Again, because each node knows the location of its private buffers at every other node, it keeps track of exactly where to write the memories of remote nodes. The sender of a file then writes the file data to the large buffer and the file information to the small buffer. The file information includes a pointer to the beginning of the file in the large buffer. Again, polling by the receiver is done on the sequence numbers at the end of the server loop. When the receiver detects the arrival of a file, it copies it to another buffer and sends it back to the requesting client.

Version RDMA also improves on version RDMA-1c by having the receiver of file data send the data to the client right out of the large communication buffer. By doing this, version RDMA eliminates the expensive data copy made at the receiver of a file transfer message.

Finally, version Zero improves on version RDMA by eliminating the data copy at the transmitter of a file transfer message. For that, all pages containing cached files are registered in VIA.

Fig. 4 depicts the throughput increase of the different versions of PRESS for each trace on our 8-node server cluster. The throughput increases were computed with respect to version Base. Table 2 lists the number of messages and bytes that each version transfers for the Clarknet trace as an example.

The figure shows that Version RDMA-1c does not significantly improve performance, even though interrupts

TABLE 2
Intracluster Communication, RMW, and Zero-Copy for Clarknet

Version	Msg type	Num msgs per 1K reqs	Num Kbytes per 1K reqs	Avg msg size (bytes)
Base	Flow	435.3	1.7	4.0
	Forward	766.7	40.3	55.1
	Caching	0.8	0.1	59.1
	File	766.7	6646.9	9091.2
	TOTAL	1969.5	6689.0	–
RDMA-1c	Flow	1362.0	17.4	13.1
	Forward	778.2	45.0	59.2
	Caching	0.0	0.0	N/A
	File	778.2	6815.7	8968.0
	TOTAL	2918.4	6878.1	–
RDMA	Flow	1475.0	18.9	13.1
	Forward	842.9	48.6	59.1
	Caching	0.0	0.0	N/A
	File	842.9	7417.8	9012.0
	TOTAL	3160.8	7485.3	–
Zero	Flow	1477.1	18.9	13.1
	Forward	844.1	48.8	59.2
	Caching	0.002	0.0001	63.1
	File	844.1	7433.3	9018.0
	TOTAL	3165.3	7501.0	–

for message reception and the receive thread itself have been eliminated. The reason for this surprising result is that using remote memory writes requires two messages per file (one with the file data and one with metadata/buffer management information), rather than a single message. The effect of this increase in the number of messages can be observed by comparing the number of flow control messages and the total number of messages in versions Base and RDMA-1c in Table 2.

In contrast, versions RDMA and Zero do exhibit performance benefits. The benefits from version RDMA range from 11 percent for WC98 to 23 percent for Rutgers, averaging 15 percent. Version Zero outperforms all other versions, reaching improvements that range from 15 percent for WC98 to 29 percent for Rutgers and average 20 percent. The gains achieved by these versions stem from avoiding large copies.

Overall, by comparing version Zero against the TCP-only implementation, we find that user-level communication improves throughput by up to 52 percent. On average, the performance gains are 43 percent. No specific property of this type of communication is responsible for the vast majority of the gains. Low processor overhead is responsible for about 19 percent, remote memory writes for file transfers are responsible for about 18 percent, and zero-copy file transfers are responsible for about 6 percent. *Note that remote memory writes have commonly been studied in the literature in scenarios where they obviate the need for data copies at receivers, so we credit the gains achieved by version RDMA to remote memory writes and the gains of version Zero to zero-copy transfers.*

4.3 Comparison against Other Servers

In this section, we compare the performance of PRESS to that of hand-off and traditional servers. In all experiments, we made an effort to define the node overload threshold (parameter T , as described in Section 3.1) such that all

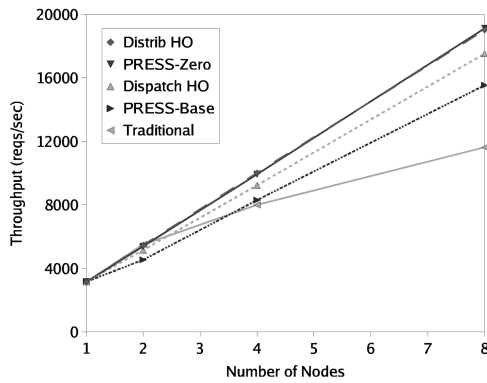


Fig. 5. Throughputs for the Clarknet trace.

servers forwarded roughly the same percentage of requests. With this approach, we guarantee that the differences between systems are dominated by their intracluster communication properties.

Our throughput comparison results are shown in Fig. 5 and Fig. 6. As a representative example of the scalability of the different systems, Fig. 5 plots the throughput of each system for the Clarknet trace as a function of the number of cluster nodes. The figure presents results for the locality-oblivious server (labeled “Traditional”), the base version of PRESS (“PRESS-Base”), the dispatcher-based hand-off server (“Dispatch HO”), the Zero version of PRESS (“PRESS-Zero”), and the distributed hand-off server (“Distrib HO”). Fig. 6 plots the throughput of each system on eight nodes for each of our traces.

Many interesting observations can be made from these figures. They show that the locality-conscious servers are very efficient and scalable compared to their locality-oblivious counterpart. The performance advantage of PRESS over the locality-oblivious server on eight nodes is 77 percent on average: 173 percent for Rutgers, 91 percent for WC98, 64 percent for Clarknet, 38 percent for Usask, and 18 percent for Nasa.

The figures also show that PRESS-Zero and the distributed hand-off server perform virtually the same, showing that the extra overhead of transferring file data within the cluster in PRESS does not degrade performance when user-level communication is exploited to the fullest. As we illustrated in Fig. 4, PRESS performs significantly

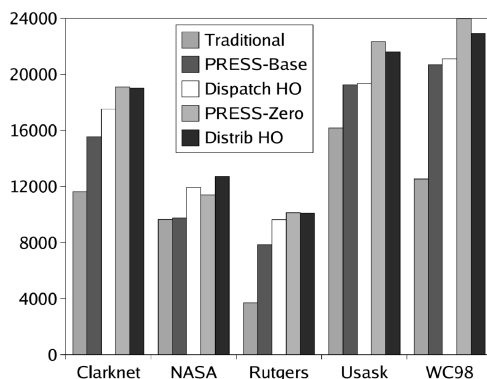


Fig. 6. Throughputs on eight nodes.

TABLE 3
Main Execution Statistics on Eight Cluster Nodes

Logs	Trad		PRESS-Zero			Distributed HO		Dispatcher HO		
	HR	LHR	THR	Fwd	LHR	THR	Fwd	LHR	THR	Fwd
Clarknet	98.8	99.6	99.9	87.4	99.6	99.9	82.0	99.6	99.9	85.7
Nasa	98.3	91.2	98.7	85.6	94.6	98.7	75.1	94.4	98.7	77.3
Rutgers	96.4	94.1	99.2	86.8	94.1	99.2	86.9	94.7	99.2	85.0
Usask	98.6	99.9	100	87.5	99.9	100	72.5	99.9	100	82.1
WC98	98.4	99.8	99.9	87.4	99.8	99.9	86.0	99.8	99.9	85.4

Dispatcher runs by itself on an extra node. All values are percentages.

worse when remote memory writes and zero-copy transfers are not used.

The dispatcher-based server performs slightly worse than its distributed counterpart, even though the internode communication bandwidth is plentiful and the dispatcher never becomes a bottleneck in our experiments. This slight performance degradation is caused by the larger number of messages involved in consulting the dispatcher on all requests.

The statistics that explain these results are presented in Table 3. From left to right, the table shows the average cache hit rate of the locality-oblivious server and several statistics for PRESS and the hand-off servers on eight nodes: the average cache hit rate of all requests (coming from either clients or remote nodes) serviced locally by each node (“LHR”), the average total cache hit rate (“THR”), and the percentage of forwarded requests (“Fwd”) for each of our traces. All of these statistics correspond to executions on eight nodes, except for the executions of the dispatcher-based hand-off server, which involved nine nodes. Note that all values in the table are percentages.

The main reason for PRESS to outperform the traditional server is that by efficiently distributing requests based on locality and load balancing (approximately 87 percent of the requests are forwarded within the cluster), PRESS can achieve a higher hit rate. Considering locality in PRESS cuts down the number of disk accesses by a factor of up to 11 (Usask). The hit rate statistics of PRESS and the hand-off servers are similar given that we adjusted the amount of request forwarding with this purpose.

4.4 Discussion

In summary, we find that all aspects of user-level communication are important for content-aware servers such as PRESS. Furthermore, we find that user-level communication enables PRESS to achieve similar performance to hand-off servers, which so far have been considered more efficient than request-reply servers.

Extrapolating from experiments. Our study was performed in a scenario where the nodes’ CPUs (in reality, the nodes’ memory bandwidths) are the performance bottleneck, regardless of cluster size. When main memories are relatively small, the disk subsystem becomes the bottleneck. When communication bandwidth is relatively low, the network subsystem becomes the bottleneck. Under these other scenarios, user-level communication brings no performance benefits since the CPUs are underutilized and can thus withstand the overhead of kernel-level communication.

In terms of the comparison between servers, memory sizes are also a factor. Larger memories improve the

performance of the traditional server much more significantly than that of the locality-conscious servers. Nevertheless, the locality-conscious servers outperform the traditional server until each memory becomes large enough to almost contain a trace's entire working set. When that is the case, all systems except for the dispatcher-based server behave in the same way. The dispatcher-based server involves communication between server nodes and the dispatcher, regardless of the memory size. The number of hand-offs does decrease with larger memory, though.

The amount of the internal communication bandwidth affects the comparison between hand-off and request-reply servers. When this bandwidth is relatively low, the latter servers have a clear disadvantage for transferring file data inside the cluster.

Given the characteristics of our experimental environment, it is clear that the locality-conscious servers we studied should scale well for larger cluster sizes. In particular, the PRESS systems exhibit scalable request distribution overheads, so they should scale well up to the point that they saturate the interconnection network. We can determine the saturation point by extrapolating the network utilizations found for eight nodes. For the trace with the highest network utilization (NASA), this point would be around 32 nodes. For the trace with the lowest network utilization (WC98), saturation would happen at about 80 nodes. If we had a separate network for external communication, these saturation points would be around 64 and 160 nodes, respectively. Further scaling would require more communication bandwidth, i.e., a higher performance or additional network. In contrast, Dispatch HO should scale well but only until the dispatcher becomes a bottleneck. In a previous paper [12], we used a validated analytical model to show that request-reply and distributed hand-off servers indeed scale very well.

Request latencies. Although our focus has been on throughput, we also measured request latencies at the clients when the servers are run at maximum throughput. These results show that PRESS also performs well in terms of latencies. For example, PRESS-Zero achieves the same average request latency as Distrib HO (27 milliseconds) for Clarknet on eight nodes. In comparison, Dispatch HO (30 milliseconds), PRESS-Base (33 milliseconds), and Traditional (43 milliseconds) achieved higher average latencies on the same configuration. The other traces exhibit the same trends. However, as we mentioned before, the latency differences between the servers are small relative to the hundreds of milliseconds that clients would perceive in a wide-area deployment.

Implications for other servers. Our findings have implications beyond PRESS. Communication in PRESS and other portable content-aware servers is so intensive that it is unlikely that other servers could accrue more significant improvements from user-level communication. Furthermore, even though we focus on servers with a content-aware request distribution and PRESS in particular, our observations should directly extend to other types (such as ftp, e-mail, proxy, or file) and implementations of cluster-based servers as long as files or file blocks are effectively transferred among the cluster nodes. A few

examples of servers in this class are the Swala WWW server [19], the Porcupine e-mail server [29], and servers based on the Direct Access File System (DAFS) [21].

5 RELATED WORK

User-level communication and VIA. Previous studies of user-level communication were performed either for microbenchmarks [1], [7], [9], [10], [15], [16], [27], [30], [35], [36] or in the context of scientific applications [22], [28], [31]. This paper extends those studies to real, nonscientific applications, in particular to the now very popular cluster-based servers.

Both industry and academia have produced implementations of VIA [4], [9], [10], [17], [30], [37]. These studies have been performed in the context of microbenchmarks, parallel, and/or distributed applications. The exceptions are DAFS [21] and the work of Zhou et al. [38]. DAFS is an emerging standard for network-attached storage that takes advantage of user-level communication standards. Its current implementation uses VIA over cLAN. Zhou et al.'s work uses VIA to improve I/O path performance between a database server and its storage subsystem. The benefits of the different features of VIA were not addressed explicitly in these works. Our paper extends the small body of work on this industry standard, while isolating the performance benefits of several of its main features (low overhead, remote memory write, copy avoidance) for cluster-based network servers.

InfiniBand has also recently been evaluated using microbenchmarks and parallel scientific applications on top of MPI [20]. Our study and the trends it illustrates should extend directly to InfiniBand-based server clusters.

Request distribution in locality-conscious servers. Pai et al. [25] proposed the first locality-conscious network server. In their system, the request distribution was performed by a content-aware front-end node, which handed off the TCP connections to the back-end servers. To eliminate the front-end bottleneck, Aron et al. [3] proposed a dispatcher-based hand-off server. In their system, the local servers communicate with the dispatcher on all requests to find out which servers should actually service them. As we mentioned before, our implementation of this approach suffered from the excessive number of dispatcher-related messages. The implementation that we studied in this paper is an optimization of their server.

In a previous paper [6], we modeled the potential benefits of fully distributed locality-conscious hand-off servers analytically and proposed one such server, called L2S. The evaluation of L2S was done in simulation. The fully distributed hand-off server that we experimentally studied in this paper is inspired by L2S.

Aron et al. [2] studied the performance of different schemes for supporting persistent TCP connections that use a locality-conscious front-end node. The problem they tackled is that requests in a single connection may have to be assigned to different cluster nodes according to the request distribution policy. They considered two schemes for handling persistent TCP connections while retaining locality: 1) multiple hand-off—the front-end node may hand off the connection multiple times (to different back-end nodes) depending on the locality of each request—and

2) back-end forwarding—the front-end hands off each connection to a single back-end node based on the first request to arrive on the connection. After this, the front-end informs the connection handling node about which other back-end node should handle each request. The connection handling node then forwards the request to the other node, receives its reply, and, finally, forwards the reply to the client. Their worst-case simulation results show that back-end forwarding performs better than multiple hand-off for requested files averaging up to 6-12 KBytes, while the converse is true for larger average requested file sizes. Simulation results with a real workload show that back-end forwarding performs almost as well as the significantly more complex multiple hand-off strategy.

Even though we did not experiment with persistent connections, one could loosely equate their multiple hand-off and back-end forwarding servers with our hand-off and request-reply servers, respectively. However, several assumptions underlying the two studies are quite different. Aron et al. assumed that the request distribution algorithm, back-end forwarding, and hand-off were all completely implemented within the kernel, whereas we always assume a user-level distribution algorithm and user-level to user-level forwarding. Furthermore, our comparison of distribution strategies relies on user-level communication over a much faster network. Finally, we considered the impact of the features of a sophisticated user-level communication substrate. As a result of these differences, we find that the basic version of our request-reply server performs substantially worse than both of our hand-off servers for a wide variety of real traces and average requested file sizes. However, when user-level communication is exploited to the fullest, our request-reply server consistently performs better than the dispatcher-based hand-off server and virtually the same as the fully distributed hand-off server.

PRESS. Although we did not focus on availability in this paper, PRESS can detect and recover from node and server crashes. In addition, new server nodes can be added to or removed from the cluster dynamically. The effect of the intracluster communication protocol on the availability of PRESS has been studied in [23].

The server that is closest in spirit to PRESS is the Porcupine e-mail server [29]. Porcupine shares two important characteristics of PRESS: 1) It does not rely on TCP hand-offs or any other operating system extensions and 2) it considers both locality and load balancing in distributing requests. The key difference between this system and PRESS is that Porcupine's request distribution strategy focuses on *disk* locality and *disk* load, rather than *cache* locality and *node* load. This focus probably stems from the properties of e-mail workloads, whereas partitioning disk locality information across user managers rather than replicating it at all nodes probably seeks to reduce network traffic (at the expense of higher service latency). These two design decisions effectively mean that Porcupine's dissemination of information across the cluster does not have to be as aggressive as in PRESS, which relies on high-performance communication to make request distribution decisions that are as accurate as possible.

One of the key goals of PRESS is to achieve a good measure of portability. However, portability in cluster-

based network servers is also achievable in ways that differ from our internode communication approach. Four common approaches are to use:

1. Round-Robin DNS only,
2. content-oblivious front-ends only,
3. HTTP redirection, and
4. a content and/or locality-aware front-end.

The first approach leads to imbalanced loads and does not allow for considering cache locality or intelligently assigning requests to nodes. The second approach can accurately balance load, but suffers from the latter problem as well. The third approach allows for considering both locality and load balancing, but is not transparent to clients and increases response time significantly. Finally, the fourth approach has limited scalability since the front-end has to inspect all requests and relay all replies. Nevertheless, a high-throughput locality-aware front-end should be able to perform as well as PRESS and the distributed hand-off server for the small cluster in our experiments.

6 CONCLUSIONS

In this paper we proposed a novel content-aware WWW server. We also quantified the impact of user-level communication on the performance of our server. Our results demonstrated that processor overhead, remote memory writes, and zero-copy transfers can all provide nontrivial performance gains. Finally, we compared our server to systems that use a potentially more efficient request distribution approach, but rely on fairly complex operating system extensions. Our results demonstrated that user-level communication can mitigate any performance degradation in content-aware servers.

These conclusions are important in that they encourage future user-level communication substrates to continue to provide a full set of features. Furthermore, our conclusions suggest that network service providers do not have to invest in servers that require kernel-level infrastructure development just for performance. In fact, another study of our server [23] suggests that availability is another reason why relying on user-level communication is a good idea.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation under grants CCR-0100798 and EIA-9986046.

REFERENCES

- [1] S. Araki, A. Bilas, C. Dubnicki, J. Edler, K. Konishi, and J. Philbin, "User-Space Communication: A Quantitative Study," *Proc. Supercomputing*, Nov. 1998.
- [2] M. Aron, P. Druschel, and W. Zwaenepoel, "Efficient Support for P-HTTP in Cluster-Based Web Servers," *Proc. USENIX Ann. Technical Conf.*, June 1999.
- [3] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, "Scalable Content-Aware Request Distribution in Cluster-Based Network Servers," *Proc. USENIX Ann. Technical Conf.*, June 2000.
- [4] M. Banikazemi, V. Moorthy, L. Herger, D.K. Panda, and B. Abali, "Efficient Virtual Interface Architecture Support for the IBM SP Switch-Connected NT Clusters," *Proc. 14th Int'l Parallel and Distributed Processing Symp.*, 2000.

- [5] T. Berners-Lee, R. Fielding, and H. Frystyk, "RFC 1945: HyperText Transfer Protocol—HTTP/1.0," technical report, HTTP Working Group, May 1996.
- [6] R. Bianchini and E.V. Carrera, "Analytical and Experimental Evaluation of Cluster-Based WWW Servers," *World Wide Web J.*, vol. 3, no. 4, pp. 215-229, Dec. 2000.
- [7] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg, "Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer," *Proc. 21st Ann. Int'l Symp. Computer Architecture*, pp. 142-153, Apr. 1994.
- [8] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.-K. Su, "Myrinet: A Gigabit per Second Local Area Network," *IEEE Micro*, vol. 15, no. 1, pp. 29-36, Feb. 1995.
- [9] P. Bozeman and B. Saphir, "A Modular High Performance Implementation of the Virtual Interface Architecture," *Proc. Second Extreme Linux Workshop*, June 1999.
- [10] P. Buonadonna, A. Geweke, and D. Culler, "An Implementation and Analysis of the Virtual Interface Architecture," *Proc. Supercomputing*, Nov. 1998.
- [11] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Yu, "The State of the Art in Locality Distributed Web-Server Systems," Technical Report RC22209, IBM, Oct. 2001.
- [12] E.V. Carrera and R. Bianchini, "Efficiency vs. Portability in Cluster-Based Network Servers," *Proc. Eighth Symp. Principles and Practice of Parallel Programming*, pp. 113-122, June 2001.
- [13] E.V. Carrera, S. Rao, L. Iftode, and R. Bianchini, "User-Level Communication in Cluster-Based Servers," *Proc. Eighth Int'l Symp. High-Performance Computer Architecture*, pp. 275-286, Feb. 2002.
- [14] Compaq Corp., Intel Corp., and Microsoft Corp., *Virtual Interface Architecture Specification, Version 1.0*, 1997.
- [15] C. Dubnicki, L. Iftode, E. Felten, and K. Li, "Software Support for Virtual Memory-Mapped Communication," *Proc. 10th Int'l Parallel Processing Symp.*, Apr. 1996.
- [16] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A.M. Merritt, E. Gronke, and C. Dodd, "The Virtual Interface Architecture," *IEEE Micro*, vol. 18, no. 2, pp. 66-76, Mar. 1998.
- [17] Emulex, *cLAN Cluster Switch*, 2001.
- [18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616: HyperText Transfer Protocol—HTTP/1.1," technical report, HTTP Working Group, June 1999.
- [19] V. Holmedahl, B. Smith, and T. Yang, "Cooperative Caching of Dynamic Content on a Distributed Web Server," *Proc. Seventh IEEE Int'l Symp. High Performance Distributed Computing*, pp. 243-250, July 1998.
- [20] J. Liu, J. Wu, S. Kinis, P. Wyckoff, and D.K. Panda, "High Performance RDMA-Based MPI Implementation over InfiniBand," *Proc. 17th Ann. ACM Int'l Conf. on Supercomputing*, June 2003.
- [21] K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J.S. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, and E. Gabber, "Structure and Performance of the Direct Access File System," *Proc. USENIX Ann. Technical Conf.*, June 2002.
- [22] R. Martin, A. Vahdat, D. Culler, and T. Anderson, "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture," *Proc. 24th Int'l Symp. Computer Architecture*, June 1997.
- [23] K. Nagaraja, N. Krishnan, R. Bianchini, R.P. Martin, and T.D. Nguyen, "Evaluating the Impact of Communication Architecture on the Performance of Cluster-Based Services," *Proc. Ninth Symp. High Performance Computer Architecture (HPCA-9)*, Feb. 2003.
- [24] Foundry Networks, "ServerIron," <http://www.foundrynetworks.com>, May 2003.
- [25] 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," *Proc. Eighth ACM Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 205-216, Oct. 1998.
- [26] V.S. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An Efficient and Portable Web Server," *Proc. USENIX Ann. Technical Conf.*, June 1999.
- [27] S. Pakin, M. Karamcheti, and A. Chien, "Fast Messages (FM): Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors," *IEEE Parallel and Distributed Technology*, vol. 5, no. 2, pp. 60-73, Apr. 1997.
- [28] M. Rangarajan and L. Iftode, "Software Distributed Shared Memory over Virtual Interface Architecture: Implementation and Performance," *Proc. Third Extreme Linux Workshop*, 2000.
- [29] Y. Saito, B.N. Bershad, and H.M. Levy, "Manageability, Availability and Performance in Porcupine: A Highly Scalable, Cluster-Based Mail Service," *Proc. 17th ACM Symp. Operating Systems Principles*, Dec. 1999.
- [30] E. Speight, H. Abdel-Shafi, and J.K. Bennett, "Realizing the Performance Potential of the Virtual Interface Architecture," *Proc. Int'l Conf. Supercomputing*, pp. 184-192, June 1999.
- [31] R. Stets, S. Dwarkadas, L. Kontothanassis, U. Rencuzogullari, and M. Scott, "The Effect of Network Total Order, Broadcast, and Remote-Write Capability on Network-Based Shared Memory Computing," *Proc. Sixth Int'l Symp. High-Performance Computer Architecture*, Jan. 2000.
- [32] W. Tang, L. Cherkasova, L. Russell, and M. Mutka, "Modular TCP Handoff Design in Streams Based TCP/IP Implementation," *Proc. First Int'l Conf. Networking*, pp. 71-80, July 2001.
- [33] H. Tesuka, A. Hori, and Y. Ishikawa, "PM: A High-Performance Communication Library for Multi-User Parallel Environments," Technical Report TR-96015, Real World Computing Partnership, Nov. 1996.
- [34] The InfiniBand Trade Assoc., *InfiniBand 1.0 Specifications*, <http://www.infinibandta.org>, Oct. 2000.
- [35] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," *Proc. 15th ACM Symp. Operating Systems Principles*, pp. 40-53, Dec. 1995.
- [36] K.G. Yocum, J.S. Chase, A.J. Gallatin, and A.R. Lebeck, "Cut-Through Delivery in Trapeze: An Exercise in Low Latency Messaging," *Proc. IEEE Symp. High-Performance Distributed Computing*, Aug. 1997.
- [37] K.H. Yum, E.J. Kim, and C.R. Das, "QoS Provisioning in Clusters: An Investigation of Router and NIC Design," *Proc. 28th Ann. Int'l Symp. Computer Architecture*, June 2001.
- [38] Y. Zhou, A. Bilas, S. Jagannathan, C. Dubnicki, J.F. Philbin, and K. Li, "Experiences with VI Communication for Database Storage," *Proc. 29th Int'l Symp. Computer Architecture*, May 2002.
- [39] H. Zhu, B. Smith, and T. Yang, "Scheduling Optimization for Resource-Intensive Web Requests on Server Clusters," *Proc. 11th Ann. ACM Symp. Parallel Algorithms and Architectures*, pp. 13-22, June 1999.



Enrique V. Carrera received the MSc degree in electrical engineering from the Catholic University of Rio de Janeiro in 1996 and the DSc degree in computer science from the Federal University of Rio de Janeiro in 1999. From 2000 until 2003, he was a postdoctoral associate in the Department of Computer Science at Rutgers University. He is currently an assistant professor at the University San Francisco de Quito, Ecuador. His research interests include cluster-based network servers, parallel and distributed applications, and mobile network services. He is a member of the IEEE and the ACM.



Ricardo Bianchini received the PhD degree in computer science from the University of Rochester in 1995. From 1995 until 1999, he was an assistant professor at the Federal University of Rio de Janeiro, Brazil. He is now an associate professor in the Department of Computer Science at Rutgers University. His current research interests include the performance, availability, and energy consumption of cluster-based network servers. He has received several awards, including the US National Science Foundation CAREER award. He is a member of the IEEE, the IEEE Computer Society, and the ACM.