Efficiency vs. Portability in Cluster-Based Network Servers

Enrique V. Carrera and Ricardo Bianchini

Department of Computer Science
Rutgers University
{vinicio,ricardob}@cs.rutgers.edu

Technical Report DCS–TR–427, November 2000

Abstract

Efficiency and portability are usually conflicting objectives for cluster-based network servers that distribute the clients' requests across the cluster based on the actual content requested. Our work is based on the observation that this efficiency vs. portability tradeoff has not been discussed before in the literature. To fill this gap, in this paper we study this tradeoff in the context of an interesting class of content-based network servers, the locality-conscious servers, using modeling and experimentation. Our analytical model gauges the potential performance benefits of portable and non-portable locality-conscious request distribution with respect to a traditional, locality-oblivious server, as a function of multiple parameters. Based on our experience with the model, we design and evaluate a portable, locality-conscious server. Experiments with our server, a non-portable server, and a traditional server validate and confirm our modeling results under several real workloads. Based on our modeling and experimental results, our main conclusion is that portability should be promoted in cluster-based network servers with low processor overhead communication, given its relatively low cost ($\leq$ 15%) in terms of efficiency. For clusters with high processor overhead communication, efficiency should be the overriding concern, as the cost of portability can be very high (as high as 98% on 32 nodes). We also conclude that user-level communication can be useful even for non-scientific applications such as network servers.

1 Introduction

The number of Internet users has increased rapidly over the last few years; this growth can be expected to continue for the foreseeable future. With the emergence of the Internet computing model, where a large amount of information is stored on remote servers and accessed by clients across the Internet, this rapidly growing client base is placing significant stress on the computing resources of popular content providers and on-line stores. As a result, popular service providers have no alternative but to utilize either large multiprocessor or distributed network servers in order to achieve reasonable quality of service levels. We are interested in distributed servers (cluster-based servers in particular), as they have the potential to deliver extremely high performance at low cost. The commercial HotBot search engine is an interesting example of a popular network service that performs several million queries per day using a very large cluster-based server [15]. Several other popular services such as Yahoo! and the Google search engine are also supported by large cluster-based servers. Given the amount of traffic imposed on these services and the importance of the quality of the users' experience with the services, it is critical to understand and optimize their performance under heavy load.

Two of the key performance issues in cluster-based network servers are how the clients' requests are distributed across the different nodes of the cluster and how replies are sent back to clients. Several cluster-based servers (e.g. [10, 18, 12, 4, 23]) distribute requests based solely on a load metric, which is usually the resource utilization or the number of open connections being handled by each node. In these servers the cluster node that accepts a request (usually a TCP connection) is also responsible for serving the request (independently of other nodes) and replying with the content requested to the corresponding client.

Other cluster-based servers (e.g. [18, 24, 25, 28, 26, 20, 2, 8, 5]) can distribute requests based on the actual content requested; for instance, requests for images may be directed to a different set of nodes than requests for html files, requests for unreplicated disk data can be directed to the node that stores the data, or requests for certain files may be directed to the nodes that cache those files in memory. In these servers the cluster node that initially accepts a request (establishing a TCP connection with the client) and determines the content requested may not be the node that will actually service the request. In this case, 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 [20] 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 however, operating systems currently do not provide TCP hand-off mechanisms, so implementing hand-offs requires modifications to the operating system kernel. Thus, TCP hand-off is not portable across different operating systems or even across different versions of the same operating system. This portability problem severely limits the usefulness of these servers, as upgrading the cluster software or implementing the server on a different cluster platform requires a serious development

effort. Hereafter, we refer to servers that rely on TCP hand-off as *non-portable* servers.

The alternative 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 promotes portability, but is less efficient than using TCP hand-off. The extent of the performance degradation caused by promoting portability depends mainly on the performance of the inter-node communication subsystem and on the average size of the replies. Hereafter, we refer to servers that do not rely on TCP hand-off as *portable* servers.

Our work is based on the observation that the efficiency vs. portability tradeoff presented by these two approaches to sending replies to clients has not been discussed before in the literature. To fill this gap, in this paper we study this tradeoff in the context of an interesting class of content-based network servers, the locality-conscious servers [20, 2, 8, 5].

Locality-conscious servers are inspired by work on cooperative caching [11, 17, 13]. These servers use the set of memories of the cluster as a single large cache and distribute requests for read-mostly 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 email services, given how quickly their working sets can grow. Even though we focus on locality-conscious servers, we believe that our study should apply to any type of content-based server.

We use analytical modeling and experimentation to address this tradeoff. We use analytic modeling to gauge the potential throughput[1] benefits of both portable and non-portable locality-conscious request distribution with respect to a traditional server, as a function of the average size of the files requested, the cache hit rate, the inter-node communication performance, and the number of cluster nodes. We model these types of distribution using an open queuing network. The model is solved by instantiating its parameters and mathematically solving the system of equations that describes it. In effect, the model places an *upper bound* on the throughput achievable by portable and non-portable locality-conscious request distribution.

Our main modeling results show that for clusters that combine a high-performance system-area network (SAN) with low-overhead user-level communication (hereafter referred to as *fast-communication clusters*), portable locality-conscious distribution can increase throughput with respect to a traditional server by up to 8-fold, depending on the average size of the files requested and on the cache hit rate. However, for certain parameter combinations, a locality-conscious server on a fast-communication cluster can perform up to 8% worse than its traditional counterpart. The comparison between non-portable and portable locality-conscious servers shows that the throughput advantage of the former is never greater than 15% on fast-communication clusters of up to 32 nodes. In clusters where communication is effected by the operating system kernel over lower-performance networks (hereafter referred to as *slow-communication clusters*), portability can cost significantly more in terms of throughput; as much as 98% on 32 nodes. These results suggest that portable locality-conscious servers can be efficient and that their portability comes at relatively small performance cost for fast-communication clusters, especially given the significant gains of considering locality in the first place. The results also clearly suggest that the efficiency vs. portability tradeoff should be resolved in favor of efficiency in slow-communication clusters, as was done by other researchers [20, 2]. The main reason for this difference is the *processor overhead* involved in the inter-node communication protocol.

Although enlightening and comprehensive, these modeling results require the backing of real experiments to confirm the trends they suggest. We need to determine, for instance, how closely a real network server can approach the upper bounds generated by the model and whether the tradeoffs we see in modeling are verified in experimentation. In order to answer these and other questions and based on our experience with the model, we design and evaluate a novel portable and scalable locality-conscious server (called PRESS) for fast-communication clusters. (The modeling results favor efficiency so strongly in slow-communication clusters that we do not consider them in our experiments.)

PRESS promotes both locality and load balancing without any centralized resources, avoiding connection hand-offs by transferring a requested file back to the client through the cluster node that initially accepted the connection. To guarantee the portability of PRESS, we implemented the inter-node communication using the Virtual Interface Architecture (VIA) industry standard for user-level communication, which has been implemented for virtually all modern SANs, such as Giganet and Myrinet.

PRESS can be used to implement a variety of cluster-based network servers, such as ftp, email, or WWW servers. Our evaluation of PRESS concentrates on its application as a WWW server. The evaluation is based on its performance on an 8-node Linux cluster, 4 real WWW traces, and on a performance comparison of PRESS against a traditional server and a non-portable server, both of them based on the same code as PRESS.

Our experiments on an 8-node cluster of PCs connected by a Giganet network validate and confirm our modeling results for fast-communication clusters. In fact, all experimental results for the locality-conscious servers are within 9% on average (min=3%, max=17%) of the model predictions on 8 nodes, which shows that our model is very accurate in determining trends. Our results also show that PRESS clearly outperforms and outscales its traditional counterpart. Furthermore, the non-portable server achieves throughput that is only 7% higher on average (min=4%, max=9%) than the throughput of PRESS on 8 nodes, showing that portability is not unreasonably expensive in fast-communication clusters.

Based on our modeling and experimental results, we conclude that portability should be promoted in network servers based on fast-communication clusters, given its small cost in terms of efficiency. This conclusion is especially important to popular network service providers, which can avoid server re-implementations for each new generation of operating system or hardware platform at a small performance cost by using a portable yet efficient server such

[1] Server latencies are almost always low compared to the overall latency a client experiences establishing connections, issuing requests, and waiting for replies across a wide-area network, so we only focus on throughput.

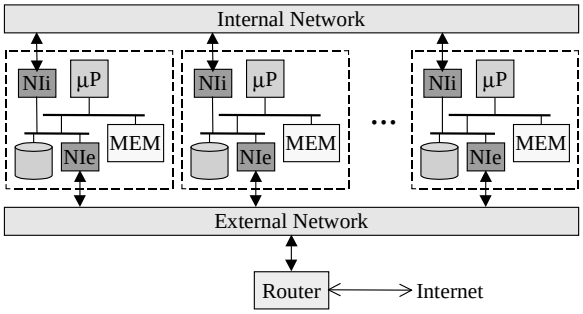

Figure 1: Typical modern cluster of workstations.

as PRESS.

We also conclude that user-level communication is the key issue that enables the implementation of cluster-based network servers that are portable and efficient at the same time. This conclusion is especially important to SAN communication protocol designers and SAN manufacturers, since the previous work on user-level communication is focused solely on scientific applications.

In summary, the main contributions made by this paper are:

- Our analytical model places upper bounds on the throughput performance of any cluster-based locality-conscious server. These bounds are used to evaluate our real servers. Furthermore, our modeling results are the first to compare portable and non-portable locality-conscious servers.
- PRESS is the first cluster-based server to distribute requests based on cache locality and load balancing considerations in a portable and scalable fashion. As far as we are aware, PRESS is the first real network server to apply user-level communication and in particular the VIA standard. Given how closely the PRESS results approximate the upper bounds set by the model (8% difference on average for 8 nodes), we find that PRESS is extremely efficient.
- Our experimental results confirm the modeling results by demonstrating that portability costs little in terms of performance in fast-communication clusters. Taken as a whole, our modeling and experimental results comprehensively address the efficiency vs. portability tradeoff we set out to study. The same trends we observed for locality-conscious servers should apply to any content-based network server.

The remainder of this paper is organized as follows. The next section describes our model in detail and presents its most important results. Section 3 presents PRESS. Section 4 describes the methodology used in our experimental study and our most important results. Section 5 discusses some related pieces of work. Finally, section 6 summarizes our findings and concludes the paper.

2 Modeling

In order to determine the potential performance benefits of locality-conscious network servers we have developed a simple open queuing network to model portable and non-portable locality-conscious servers and traditional (i.e. locality-oblivious) servers. In order to stress the communication aspects of these servers, we focus on servers of read-mostly files. In the next two subsections, we present our model and its results.

2.1 The Model

Figure 1 presents a typical architecture for a modern cluster of workstations or PCs, or a fast-communication cluster. The nodes can be connected by one or more networks. Each node is a full-blown commodity workstation or PC, with one or more processors, multiple levels of caches, main memory, one or more disks, and one or more network interfaces. Fast-communication clusters usually have two networks: an inter-node or "internal" network and an "external" network that connects the cluster to the outside world. Communication over the internal network is usually effected at the user level. In contrast, slow-communication clusters usually only have one network, which handles both inter-node and external message traffic. In these clusters, communication is usually based on kernel-level communication protocols, such as TCP and UDP. We depict the cluster with two networks and each node with one processor, one disk, and two network interfaces, because this is the architecture of our own real cluster.

Figure 2 depicts our model of a locality-conscious server running on a fast-communication cluster of N workstations. (The models for a locality-conscious server running on a slow-communication cluster and a traditional server running on any cluster are straightforward variations of the one we present.) The model assumes that all queues are M/M/1 [16]. In the model, requests for files arrive at the bridge/router with a rate $N \times \lambda$ and are routed at a rate μ_r. Assuming that the server balances the incoming load perfectly, the probability that a request is assigned to a specific workstation is the same for all the workstations ($1/N$). Thus, each external network interface receives requests with a rate λ and processes them at a rate μ_e. The initial processing of requests (reading and parsing) is done by the CPU at a rate μ_p. If the requested file is cached locally the node just replies to the request at a rate μ_m. If the file is cached only remotely, the locality-conscious server requires some mechanism for forwarding the request to another workstation, so we assume that requests are forwarded by the CPU at a rate μ_f. Each internal interface processes forwarded requests and replies at a rate μ_i. (From now on, we will refer to the node that receives a client request as the "initial" node and the node to which the request is forwarded as the "service" node.)

After a request is received by the service node, the server behavior depends on whether the file can be found in the main memory cache. If the requested file is cached, it is sent back by the CPU to the initial node through the internal cluster network with a rate μ_s. Otherwise, the node must first read the file from disk, store it in its memory, and then transfer the file across the cluster (again with a rate μ_s). The probability that a requested file is read from a specific disk is the same for all the disks ($1/N$). Thus, each disk receives read requests with a rate $(1 - H) \times \lambda$ (where H is the probability that a requested file is cached in main memory) and processes them at a rate μ_d. The CPU of the initial node receives the reply of a forwarded request at a rate μ_g. Finally, when the requested file is ready to be sent out to the client, it is sent to the bridge/router at a rate μ_e.

Other important parameters to our model are the cache (main memory) size per node (C), the average size of the requested files (S), the number of files stored on the server (F), and the file re-

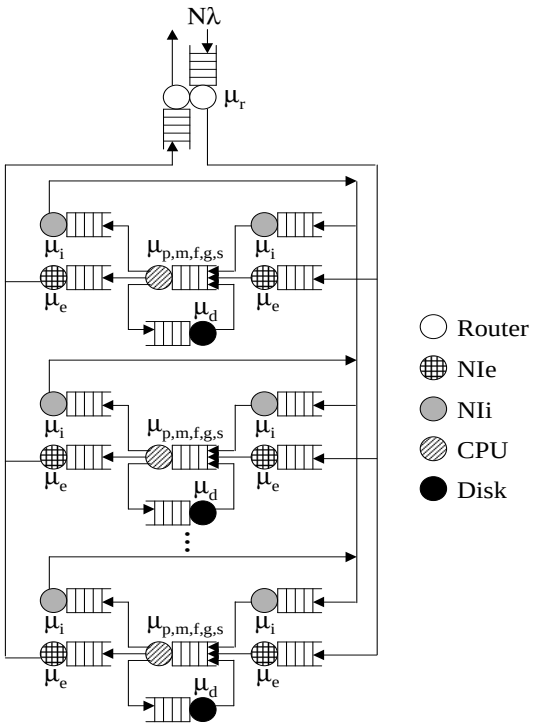

Figure 2: Model for a fast-communication cluster.

quest distribution. This latter parameter deserves further comments. In this study, we concentrate on heavy-tailed distributions of access, such as the ones exhibited by WWW servers [6]. Such distributions can be approximated by means of Zipf-like distributions [6], where the probability of a request for the i'th most popular file is proportional to $1/i^{\alpha}$ with α typically taking on some value less than unity.

With these parameters, we can define the total cache space and the average cache hit rate for the two types of servers we consider. In the traditional server, the total cache space C_{lo} is only C bytes, since the most requested files end up cached by all nodes. On the other hand, a locality-conscious server can use $C_{lc} = N \times C$ bytes of cache space if no file replication is allowed, or $C_{lc} = N \times (1 - R) \times C + R \times C$ bytes, if an R percentage of the main memory is used for file replication. (Note that we can regard a traditional server as a locality-conscious server with R = 1).

With these cache sizes, the average cache hit rate can be defined as: $H = z(n, F)$, where $z(n, F)$ represents the accumulated probability of requesting the n most accessed files in a Zipf-like distribution of the requests to F files. The number of cached files (n) is equal to $\min(C_{lo} \div S, F)$ for the traditional server, and $\min(C_{lc} \div S, F)$ for the locality-conscious server. Furthermore, the percentage of requests forwarded to another workstation in the locality-conscious server can be defined as: $Q = (N - 1) \times (1 - h) \div N$, where the hit rate for replicated files (h) is equal to $z(\min(R \times C \div S, F), F)$. Note that our use of $z(n, F)$ to describe hit rates effectively means that the most accessed files are always cached in the model.

These hit rate definitions are useful and intuitive, but in order to simplify the presentation of our results, we define the locality-conscious hit rate (H_{lc}) as a function of the traditional server's hit rate (H_{lo}) as follows: $H_{lc} = z(\min(C_{lc} \div S, f), f)$, where f is such that $H_{lo} = z(C_{lo} \div S, f)$. In the same way, we define the percentage of requests forwarded to another workstation as: $Q = (N - 1) \times (1 - h) \div N$, where $h = z(R \times C_{lo} \div S, f)$.

Note that all the above definitions assume that the probability of an unreplicated file to be cached by a specific workstation is the same for all the workstations ($1/N$). The first two columns of table 1 summarize the model parameters and their descriptions.

As our model assumes perfect load balancing and does not consider cache replacements and contention for network wires and memory and I/O buses, it provides an *upper bound on the throughput* and a *lower bound on the service latency* achievable by these servers.

Parameter Values. We carefully selected default values for our model parameters. The value of R was chosen to maximize the performance of the locality-conscious servers; recall that R has no effect on the traditional server. The service rate μ_r was selected to approximate the performance of the Cisco 7576 router (4 Gbits/s). An external network interface is present in both fast- and slow-communication clusters. The service rate of the external network interface, μ_e, was selected assuming that the interface provides 100 Mbits/s full-duplex links with 4-microsecond overhead per message. These parameters approximate the characteristics of our Fast Ethernet network interface cards. An internal network interface is only assumed for the fast-communication cluster. The service rate of the internal network interface, μ_i, was selected assuming that the interface provides 1 Gbit/s full-duplex links with 3-microsecond overhead per message. These parameters approximate the characteristics of our Giganet interface cards. The μ_d, μ_p, μ_m, μ_f, μ_s, and μ_g rates are based on measurements we took by running a single request at a time through our own real server on two cluster nodes.

The μ_s and μ_g rates have two default values, one for high-performance communication in fast-communication clusters (labeled "fast") and one for low-performance communication in slow-communication clusters (labeled "slow"). These values reflect the low processor overhead and high bandwidth achievable in a fast-communication cluster, with VIA user-level communication over Giganet, and the higher processor overhead and lower bandwidth in slow-communication clusters, with TCP/UDP kernel-based communication over Fast Ethernet for inter-node communication.

2.2 Results

In this section we study the performance of both (portable and non-portable) locality-conscious and traditional servers for both fast- and slow-communication clusters and different cluster sizes. The study concentrates on throughput as a function of the average size of requested files and the file working set size. We vary average file sizes from 2 to 64 KBytes; average sizes larger than 64 KBytes seem unrealistic for most current network servers. To simplify the analysis, we model variations in working set size using the cache hit rate of the traditional server; as the working set size increases, the cache hit rate decreases. *Note that in this section we are interested in determining trends over a wide range of parameter values; the experimental evaluation section takes care of restricting these values according to real workloads.*

Server performance on fast-communication clusters. Figures 3 and 4 show the throughput of the traditional server and that of

Param	Description	Definition or Default Value
N	Number of nodes	8
R	Percentage of replication	15%
α	Zipf constant	1
μ_r	Routing rate	512000/$size$ ops/s
μ_i	NIi transfer rate (only applicable to fast-comm cluster)	(0.000003 + $size$/64000)$^{-1}$ ops/s
μ_e	NIe transfer rate	(0.000004 + $size$/12000)$^{-1}$ ops/s
μ_p	Request read/parsing rate by CPU	9000 ops/s
μ_m	Client reply send rate (after stored locally) by CPU	(0.0001 + S/11500)$^{-1}$ ops/s
μ_d	Disk access rate	(0.0188 + S/3000)$^{-1}$ ops/s
μ_f	Inter-node request forwarding rate by CPU	60000 ops/s (portable), 55000 ops/s (non-portable)
μ_s	Inter-node reply send rate by CPU	(0.00001 + S/80000)$^{-1}$ ops/s (fast) (0.0001 + S/11500)$^{-1}$ ops/s (slow)
μ_g	Inter-node reply reception rate by CPU	(0.00001 + S/80000)$^{-1}$ ops/s (fast) (0.0001 + S/11500)$^{-1}$ ops/s (slow)
C	Total cache space *lo* = locality-oblivious *lc* = locality-conscious	$C_{lo} = C$ = 128 MBytes $C_{lc} = N \times (1 - R) \times C + R \times C$
H	Cache hit rate	$H_{lo} = z(\min(C_{lo} \div S, F), F)$ $H_{lc} = z(\min(C_{lc} \div S, F), F)$
h	Cache hit rate for replicated files	$h = z(\min(R \times C \div S, F), F)$
Q	Percentage of requests forwarded	$Q = (N - 1) \times (1 - h) \div N$

Table 1: Model parameters and their default values. S = avg file size in KBytes; $size$ = avg transfer size in KBytes.

the portable locality-conscious server, respectively. The throughput peaks represent parts of the parameter space where the workload is CPU or memory-bound, while the relatively flat areas represent parts of the parameter space where the workload is I/O-bound. We can see from the figures that the throughput of both servers increases as the average file size decreases and the cache hit rate increases. However, throughputs only increase significantly in certain areas of the parameter space.

Figure 5 directly compares the throughput of these two servers. It plots the throughput increase provided by considering locality when distributing requests, i.e. it plots the throughput results of figure 4 divided by those of figure 3. Figure 6 shows a side view of figure 5. These figures demonstrate that the portable locality-conscious server can achieve throughput that is a factor of 8 higher than that of the traditional server. The improvements start growing as the hit rate increases and the average file size decreases. However, the improvements for very small (very large) files come down quickly after the hit rate reaches about 85% (65%), as this is the point where the traditional server starts performing well. As the hit rate is increased much beyond this point, the throughput of the portable locality-conscious server becomes slightly lower than that of the traditional server (i.e. improvements become slightly smaller than 1), due to the extra cost of forwarding requests in the former server. These results suggest that considering locality can be extremely beneficial, but that a portable server should not take locality into account when the miss rate is very low and the size of files is large.

It is also important to determine how much performance we are trading for portability in a fast-communication cluster. A direct comparison between the portable and non-portable servers can provide the answer. Figures 7 and 8 plot the throughput advantage of a non-portable locality-conscious server over a portable one. The figures show that the two servers achieve the same throughput for hit rates lower than 65%, since the disks are the bottleneck in this region. As we increase the hit rates beyond this threshold, the non-portable server starts to outperform its portable counterpart due to the former system's lower inter-node communication requirements. The performance advantage of the non-portable server quickly reaches 12% for very large average file sizes but then decreases, as the high hit rate obviates the need for inter-node communication in both systems. Note that the performance advantage of the non-portable server increases with file size, from 4% for 2-KByte files to 12% for 64-KByte files. These results suggest that a portable locality-conscious server on a fast-communication cluster pays a relatively small price for its portability, especially given the significant benefits of considering locality in the first place.

Server performance on slow-communication clusters. It is important to determine the effect of the inter-node communication performance on the throughput of the different servers. To this end, figure 9 compares the throughput achievable by a portable locality-conscious server against that of a traditional server on a slow-communication cluster. Again, we divide the throughput of the locality-conscious server by that of the traditional server. Figure 10 shows a side view of figure 9.

These figures demonstrate that the portable locality-conscious server can achieve throughput that is a factor of almost 6 higher than that of the traditional server on a slow-communication cluster. However, in contrast with the results for a fast-communication cluster, taking locality into consideration can actually degrade performance more significantly (up to 38%) and over a much wider range of parameters. These results suggest that considering locality can be extremely beneficial in slow-communication clusters, but only for a restricted set of parameters.

The cost of portability is also much more significant in a slow-communication cluster. Figures 11 and 12 plot the throughput advantage of a non-portable locality-conscious server over a portable one. The figures show that the potential gains achievable by using

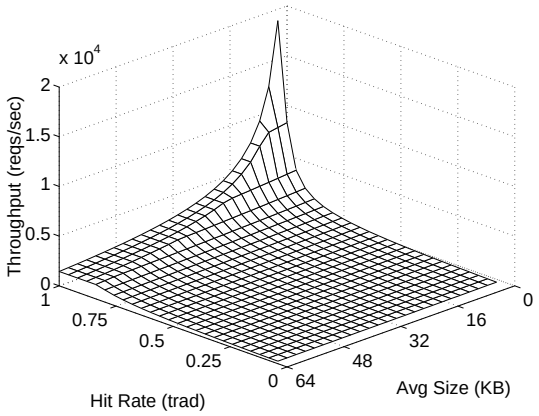

Figure 3: Throughput of a traditional server.

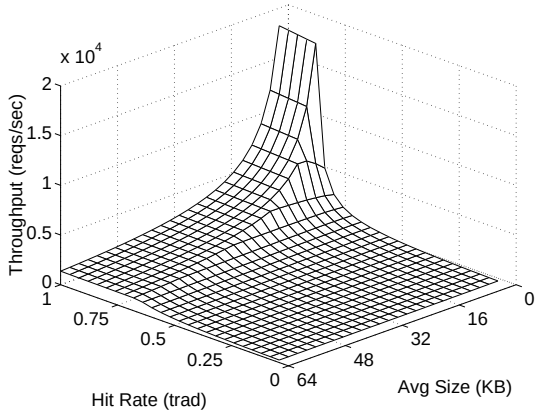

Figure 4: Throughput of a portable locality-conscious server under high-performance inter-node communication.

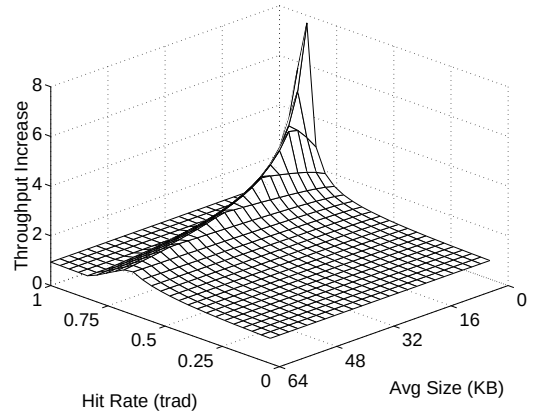

Figure 5: Throughput increase due to locality under high-performance inter-node communication.

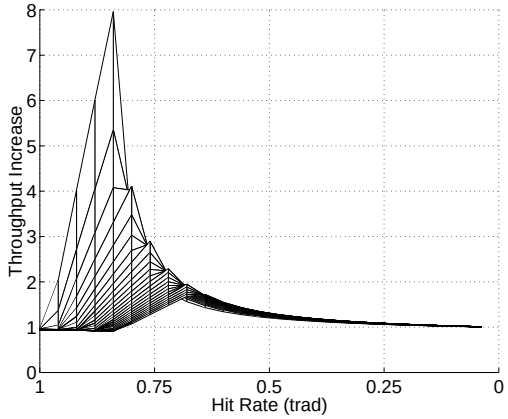

Figure 6: Throughput increase due to locality under high-performance inter-node communication (side view).

TCP hand-off can reach 79%, again for large files. In addition, the figures show that the non-portable server outperforms its portable counterpart starting at a very low (52%) hit rate.

The results for a slow-communication cluster clearly suggest that implementing TCP hand-off can provide performance gains over a large fraction of the parameter space, which confirms the intuition of researchers who implemented similar servers on these clusters [20, 2].

However, these results differ greatly from those for fast-communication clusters. There are three communication performance issues that should be considered in explaining this difference: processor overhead, network latency, and network bandwidth. Network latency is really not an issue when we consider throughput. Network bandwidth is not critical either, since the network is not a bottleneck in our parameter space, even in the case of the slow-communication cluster. *Processor overhead is the main reason why portability is inexpensive in fast-communication clusters but not in slow-communication clusters;* see the values for the μ_s and μ_g parameters. A communication substrate with high processor overhead increases processor utilization and, as a result, decreases throughput.

Effect of cluster size. The number of cluster nodes also affects the tradeoff between these different servers, as the cluster size directly influences the percentage of forwarded requests; an increase (decrease) in the number of nodes increases (decreases) the percentage of forwarding.

Figure 13 plots the maximum performance gain achievable by a portable locality-conscious server with respect to a traditional server for cluster configurations of up to 32 nodes; for larger configurations the router we model becomes a bottleneck. This figure shows that regardless of the performance of the communication substrate and the number of cluster nodes, considering cache locality can improve throughput performance by several fold.

The comparison between non-portable and portable locality-conscious servers is not as clear-cut. Figure 14 plots the maximum performance gain achievable by a non-portable locality-conscious server with respect to a portable one, again up to 32-node clusters. The figure shows that portability never costs more than 15% in terms of throughput in a fast-communication cluster. In fact, beyond 8 nodes, portability costs very little extra as we increase the cluster size. In contrast, in slow-communication clusters, portability

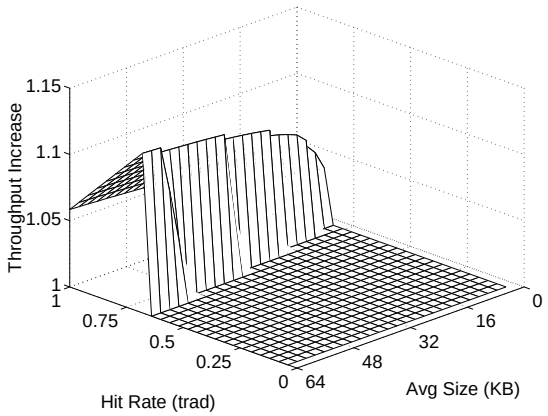

Figure 7: Throughput increase due to TCP hand-off under high-performance inter-node communication.

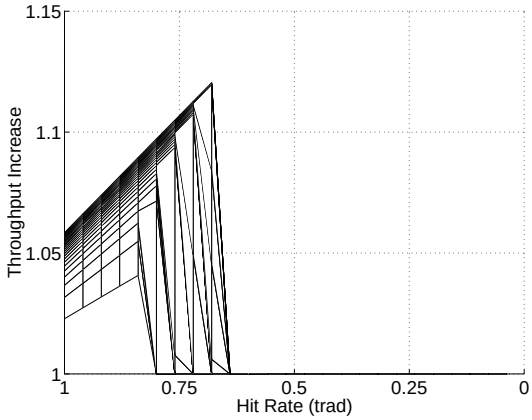

Figure 8: Throughput increase due to TCP hand-off under high-performance inter-node communication (side view).

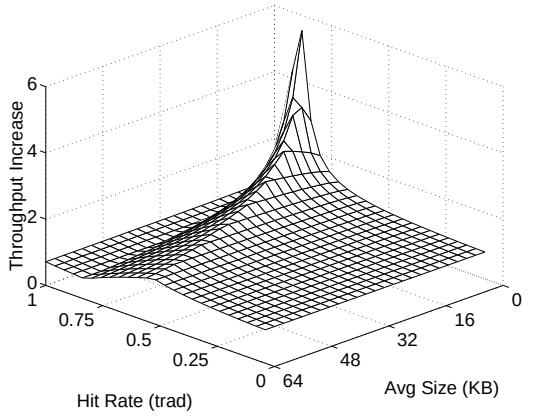

Figure 9: Throughput increase due to locality under low-performance inter-node communication.

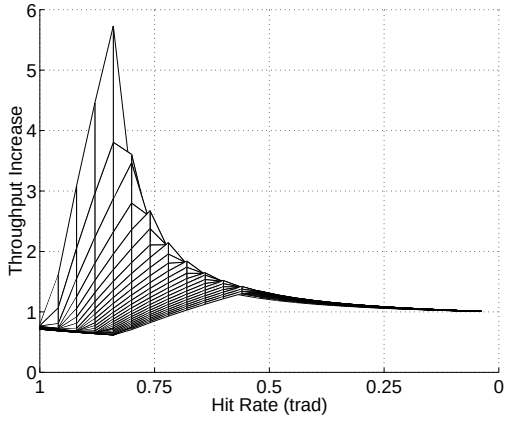

Figure 10: Throughput increase due to locality under low-performance inter-node communication (side view).

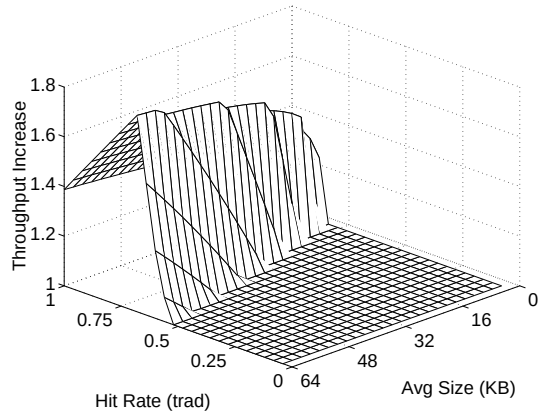

Figure 11: Throughput increase due to TCP hand-off under low-performance inter-node communication.

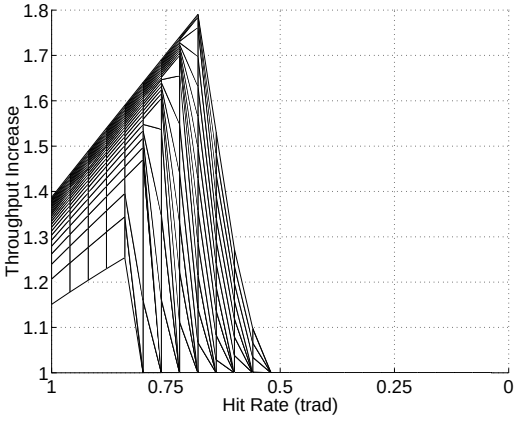

Figure 12: Throughput increase due to TCP hand-off under low-performance inter-node communication (side view).

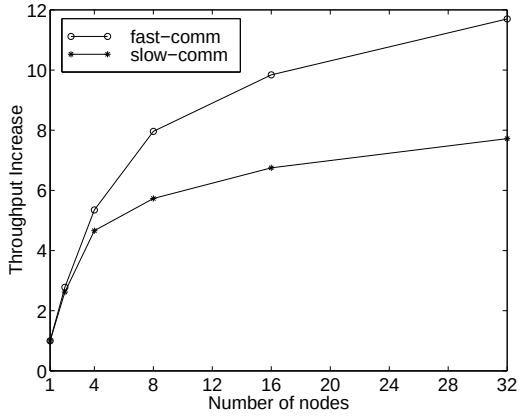

Figure 13: Portable vs. traditional: max throughput gain.

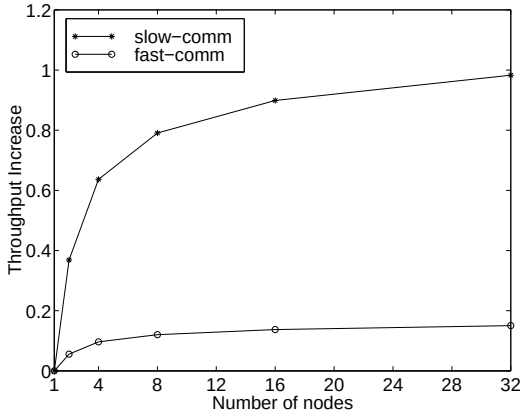

Figure 14: Non-Portable vs. portable: max throughput gain.

can cost as much as 98% on 32 nodes. These results suggest that the cost of portability does not increase significantly in larger fast-communication clusters, but does increase substantially in larger slow-communication clusters.

Given how expensive portability can be for slow-communication clusters, hereafter we focus solely on locality-conscious servers for fast-communication clusters, where the efficiency vs. portability tradeoff is more interesting.

3 PRESS

Based on the results of our modeling work, in this section we propose the Portable, Reconfigurable, Efficient, and Scalable Server (PRESS)[2] for fast-communication clusters. PRESS is based on the observation that serving a request from any memory cache is substantially more efficient than serving it from a local disk.

PRESS has two modes of operation. It starts executing in locality-oblivious mode, i.e. without communication between nodes, until it detects that the local miss rate is significant enough ($>$ 5% in our prototype) to justify taking locality into account. If that eventually happens, PRESS switches to locality-conscious mode, when each node starts to execute the request distribution algorithm described in figure 15. The idea behind it is to allow multiple nodes (named in the $Servers$ array) to cache a certain file and distribute the requests for the file among these nodes, according to load considerations. A node's load is measured as the number of connections being serviced by the node.

We assume that requests are directed to nodes using a standard method, such as round-robin DNS. When a request arrives at an initial node, the request is parsed 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 ($\geq$ 128 KBytes in our prototype) is always serviced locally by the initial node. In addition, the initial node is chosen as the service node, if this is the first time the file is requested or it already caches the requested file. If neither of these conditions holds, the least loaded node in the set of servers for the file becomes a candidate for service node. This node is chosen as a service node either when it is not overloaded

[2] This paper focuses on the portability and efficiency aspects of PRESS.

```
while (true)
  receive and parse next request r;
  if (filesize(r.target) >= S) then
     serve r but don't cache r.target;
     continue;
  if (Servers[r.target] = φ) then
     add local_node to Servers[r.target];
     n ← local_node;
  else
     if (local_node ϵ Servers[r.target]) then
        n ← local_node;
     else
        n ← {least loaded node in Servers[r.target]};
        if (n.load > T) then
           if (local_node.load < T) then
              add local_node to Servers[r.target];
              n ← local_node;
           else
              p ← {least loaded node};
              if (p.load < T) then
                 add p to Servers[r.target];
                 n ← p;
  if (n = local_node) then
     serve r;
  else
     forward r to n (reply will come later);
```

Figure 15: Pseudo-code for request distribution in PRESS.

(i.e. its number of open connections is not larger than a user-defined threshold, T = 80 in our experiments) or when it is overloaded but so are the local node and the least loaded node in the cluster. Popular files tend to force the addition of new nodes to their set of servers by overloading them. Although not shown in the figure, a node takes itself out of a server set when it replaces the corresponding file in its cache.

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 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.

It is clear then that a node trying to make a distribution decision requires locality and load information about all nodes. These data are disseminated throughout the cluster via periodic broadcasts. To avoid excessive overheads (and remain portable), internal cluster communication is implemented with VIA and the frequency of broadcasts is controlled as described below.

The dissemination of load information is simple. A node broadcasts information about its load changes when its current load is a certain number of connections (4 in our prototype) greater or smaller than the last broadcast value. Each broadcast of load information is implemented by multiple remote memory write (no data copies, implicit receive, no interrupt at receiver) VIA messages.

The dissemination of caching information is also straightforward. Whenever a node changes the server set for a certain file (i.e. it either replaces or starts caching the file), it broadcasts this information using multiple regular (one copy on each side, explicit receive operation) VIA messages. The number of these broadcasts is relatively small, since modifications to the server sets occur seldomly in steady state.

Inter-node forward and reply messages are also implemented with regular VIA messages, whereas flow control messages are implemented with remote memory writes.

Note that PRESS can be used without modification to achieve a good distribution of requests for either persistent (such as HTTP/1.1 [14]) or non-persistent (such as HTTP/1.0 [3]) connection-oriented protocols. Servers based on TCP hand-off may exhibit degraded performance for persistent connection-oriented protocols, since a connection is only handed off for the first request; a subsequent request on the same connection may have to be handled as in PRESS, if the node to which the connection was handed off is not the ideal service node.

Finally, note that our server includes several optimizations previously proposed for single-node servers (e.g. [21]). The local server at each node is event-driven but, besides the main event-processing thread, uses 4 helper threads for accessing disk, and one send thread and one receive thread for inter-node communication.

4 Experimentation

In this section we evaluate the performance of PRESS by comparing our server with a traditional (i.e. locality-oblivious) server and a non-portable server. In addition, we compare the performance of our servers with upper bounds generated by the model. Before presenting these results, we describe our experimental methodology.

4.1 Methodology

Our cluster is comprised by 8 Linux-based PCs with 750 MHz Pentium III processors, 256 MBytes of memory, a SCSI disk, and interfaces to switched Fast Ethernet and Giganet networks. The three servers we study run on this cluster. PRESS was described in detail in the previous section. Intra-cluster communication in PRESS and the non-portable server is implemented with VIA over Giganet. Under the Giganet implementation of VIA, sending a regular 4-byte message between two processes takes 19 microseconds. Out of this one-way latency, approximately 6 microseconds are spent by the communicating CPUs (3 microseconds a piece). The Giganet switch fabric peaks at 1 Gbit/s bandwidth.

Logs	Num files	Avg file size	Num requests	Avg req size	α
Clarknet	28864	14.2 KB	2978121	9.7 KB	0.77
Rutgers	18370	27.3 KB	498646	19.0 KB	0.79
Forth	11931	19.3 KB	400335	8.8 KB	0.81
Combined	64651	21.6 KB	12590736	16.2 KB	0.78

Table 2: Main characteristics of the WWW server traces.

We did not implement TCP hand-off in the Linux kernel, so our non-portable server is only an emulation of a real non-portable server. Nevertheless, our server is a very good basis for comparison, as it represents an upper bound on the throughput performance of a real non-portable server. We made a few optimistic assumptions in our emulation of the non-portable server. We emulate a TCP hand-off by delaying the initial and service processors by 2 microseconds and transferring 256 bytes over VIA/Giganet for each forward operation. The 2-microsecond processor delay at the initial node mimics the execution of a system call to retrieve the state of the TCP connection and later destroy it (without going through the TCP stack). The same delay at the service node mimics a system call to create a TCP connection (without going through the TCP three-way handshake) and set its state. The 256 bytes transferred correspond to the TCP state and the information about the file requested. The service node replies directly to the requesting client over permanent TCP connections. All the code of the non-portable server besides the forwarding operation (including the request distribution algorithm) is the same as that of PRESS.

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.

Besides our main cluster, we use another 10 Pentium-based machines to generate load for the servers. These clients connect to the cluster using TCP over the Fast Ethernet switch. They send requests to the nodes of the server in randomized fashion and reproduce four WWW server traces. Three of the traces have been used in previous studies (e.g. [1]) and are among the half-dozen server traces that are publicly available on the Internet. Clarknet is from a commercial Internet provider, Forth is from the FORTH Institute in Greece, and Rutgers contains the accesses made to the main server for the Computer Science Department at Rutgers University in the first 25 days of March 2000. We eliminated all incomplete (due to network failures or client commands) requests in the traces and ended up with the characteristics listed in table 2.

These traces cover a relatively wide spectrum of behavior. The number of files ranges from about 12000 to 28900 with an average file size between roughly 14 and 27 KBytes. The average size of the files serviced ranges from about 8 KBytes to 18 KBytes. The coefficient of the Zipf-like distribution (α) that better represents each trace also varies widely, from 0.77 to 0.81. However, these traces are

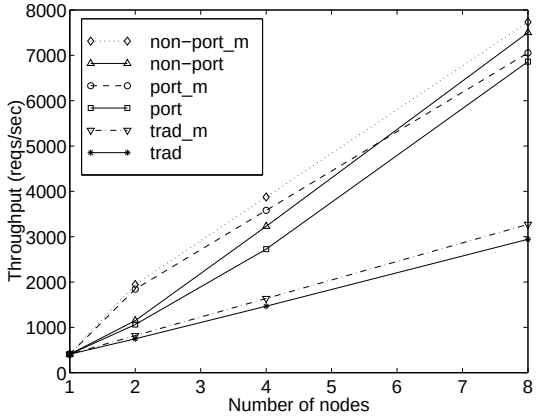

Figure 16: Throughputs for the Clarknet trace.

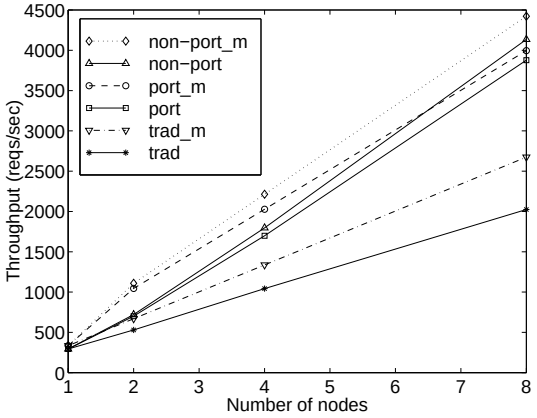

Figure 17: Throughputs for the Rutgers trace.

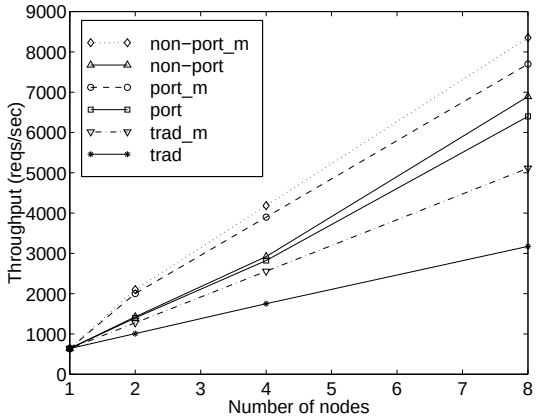

Figure 18: Throughputs for the Forth trace.

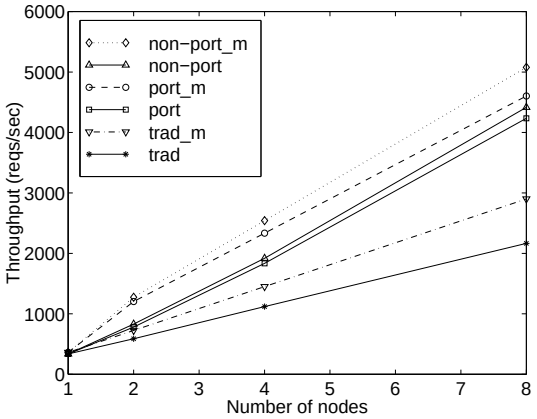

Figure 19: Throughputs for the Combined trace.

fairly old, so their working sets are very small compared to those of today's popular WWW content providers or WWW hosting services and, thus, we use small main memories (64 MBytes for Rutgers, 48 MBytes for Clarknet, and 24 MBytes for Forth) in the experiments with these traces to reproduce situations where working set sizes are significant in comparison to cache sizes. These traces and memory sizes produce single-node miss rates in the 88–93% range.

Given that traces from real WWW content providers and hosting services are not available publicly due to business reasons, we try to mimic the behavior of a hosting service with a combined trace (last line of table 2). This trace interleaves accesses from the other three traces and from another small trace, NASA [1]. The combined trace has a much larger working set size and so, for experiments involving this trace, we use 128 MBytes of main memory. The single-node miss rate of the combined trace is 89%.

We warm the node caches by executing 1/3 of the accesses in each trace before starting our measurements. To determine the maximum throughput of each system, we disregarded the timing information in the traces and made clients issue new requests as soon as possible.

Note that the range of hit rates (88–93%) and average requested file sizes (9–19 KBytes) in our experiments is an interesting one according to our model. In this region of the parameter space, the gains associated with considering locality and promoting efficiency in spite of portability are not at their peaks, but are still non-trivial, as we discuss below. Unfortunately, we cannot compare these parameters with those of real content providers and hosting services operating on the Internet today, due to an absolute lack of data on them in the literature.

4.2 Results

Base results. Our base throughput results are shown in figures 16–19. The figures plot the throughput of the non-portable (labeled "non-port"), PRESS ("port"), and locality-oblivious ("trad") servers, as well as the best possible throughput predicted by our model assuming 15% replication, as a function of the number of cluster nodes. The modeling results for each trace were computed based on the single-node cache hit rate for the trace. Model results are identified with an "_m" suffix.

Many interesting observations can be made from these figures. The figures show that the locality-conscious servers are extremely efficient and scalable, even compared to the optimal performance predicted by our model. The maximum difference between measured and modeled results for the locality-conscious servers on 8

Logs	Trad	PRESS					Non-portable				
	HR	Local HR	Total HR	% Fwd	% Load	% Cache	Local HR	Total HR	% Fwd	% Load	% Cache
Clarknet	88.9%	98.4%	99.6%	74.6%	28.9%	0.2%	98.2%	99.6%	79.2%	36.3%	0.1%
Rutgers	88.1%	89.5%	97.1%	72.0%	22.6%	0.1%	89.9%	97.1%	71.6%	30.2%	0.1%
Forth	92.8%	97.7%	99.3%	71.5%	27.4%	0.1%	97.7%	99.4%	71.8%	43.3%	0.1%
Combined	88.6%	91.1%	97.6%	73.4%	27.3%	0.5%	90.0%	97.7%	77.2%	39.3%	0.4%

Table 3: Main execution statistics on 8 cluster nodes.

nodes occurs for the Forth trace: the throughputs of both these servers for this trace are 17% lower than the corresponding modeled throughputs. On average, the difference between measured and modeled results for the locality-conscious servers on 8 nodes is only 9%. The modeled and experimental results of the locality-oblivious servers are not as close, but are always still within 38%. The main reason for this discrepancy is that the modeled miss rates are usually slightly lower than the real miss rates. This effect is more pronounced for the locality-oblivious server (and the locality-conscious servers on 2 and 4 nodes), for which workloads are more heavily disk-bound.

Another interesting set of observations is that PRESS achieves throughput that is consistently higher than that of the traditional server. The performance advantage of PRESS on 8 nodes is 51% on average; 57% for Clarknet, 48% for Rutgers, 50% for Forth, and 49% for Combined. In addition, the performance advantage of the non-portable server over PRESS is always small, 7% on average, again on 8 nodes: 9% for Clarknet, 7% for Rutgers, 8% for Forth, and 4% for Combined. These results confirm some of the conclusions we drew from our modeling results of a fast-communication cluster, showing that for the parameters of real traces: (a) locality can provide significant performance gains; (b) portability costs little in terms of performance for these clusters. Our experimental results quantify these gains and costs.

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 non-portable server: the average local cache hit rate, the average total (local plus remote) cache hit rate, the percentage of forwarded requests, the percentage of requests that lead to broadcasts of load information, and the percentage of requests that lead to broadcasts of caching information for each of our traces. All of these statistics correspond to executions on 8 nodes.

The main reason for PRESS to outperform the traditional server is that by efficiently distributing requests based on locality and load balancing (approximately 72% 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 28 (Clarknet). The messaging overhead involved in the PRESS distribution algorithm is not significant; only about 27% of the requests cause a broadcast of load information, whereas a negligible fraction of the requests cause broadcasts of caching information.

The hit rate statistics of PRESS and the non-portable server are very similar, as one would expect. However, forward messages and load broadcasts are usually more common in the non-portable server. Despite these overheads, the non-portable server does outperform PRESS in all cases due to the former server's lower communication-induced processor overhead.

Impact of important parameters. In order to fully understand the behavior of the servers we study, it is crucial to vary some of our most important parameters and observe the results. We focus on two parameters: the size of the local memories and the frequency of load information broadcasts. As one would expect, the size of the local memories affects the servers in very different ways. 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 working set. When that is the case, the three systems behave in the same way, since PRESS and the non-portable server never enter their locality-conscious mode. Regarding the frequency of load information broadcasts, we find that thresholds in the range 2-8 perform equally well. A threshold of 1 hurts performance due to an excessive number of updates, while a threshold > 8 hurts performance because load balancing is degraded.

5 Related Work

We are not aware of any other studies of the efficiency vs. portability tradeoff in content-based network servers implemented on clusters. Moreover, this work is the first to determine the full potential of portable locality-conscious request distribution. Without the support of an analytical model such as ours, it was not possible for previous researchers to determine how well their servers perform with respect to the best possible server. In this paper we compare modeling and experimental results *directly* and show that PRESS is extremely efficient.

PRESS is the first cluster-based server to distribute requests based on cache locality and load balancing considerations in a portable and fully scalable fashion. Previous cluster-based locality-conscious WWW servers were either not portable [20, 2, 8, 5], involved centralized resources [20, 2], or did not consider load balancing when distributing requests [17].

In previous papers [8, 5], we evaluated the potential benefits of non-portable locality-conscious servers and proposed a non-portable server called L2S. PRESS differs from L2S in three main ways: (1) it does not rely on TCP hand-off; (2) it strongly favors serving requests locally, since forwarded requests usually involve more overhead and the initial node must also participate in the service of forwarded requests by staging their replies; and (3) it only evicts a cached file to create space for a new file. Finally, the evaluation of L2S was based on simulations, while we evaluate a native implementation of PRESS in this paper.

The cluster-based network server that is closest in spirit to PRESS is the Porcupine e-mail server [25]. Porcupine shares two impor-

tant characteristics of PRESS: (1) it does not rely on TCP hand-offs or any other operating system modifications; and (2) it does consider 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. In more detail, when picking a node to store a message destined to a certain user, Porcupine chooses the node with the fewest pending disk operations out of the nodes that already store some part of the user's mailbox. When picking a node from which to retrieve a message, Porcupine chooses the node with the fewest pending disk operations out of the nodes that store the message. A "user manager" at each node of the cluster is responsible for maintaining the list of nodes that store the messages of a group of users. The user manager has to be consulted on email delivery and retrieval operations.

Porcupine's focus on disk load and locality probably stems from the properties of email 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. In fact, as far as we know, PRESS is the first cluster-based server to apply user-level communication and in particular the VIA standard.

User-level communication has been studied in several previous papers (e.g. [19, 9, 27, 22]). However, all of these studies were performed in the context of scientific applications or distributed shared-memory systems. Our study focuses on an important non-scientific application, network servers. In addition, although a detailed study of VIA is not our focus, this paper does extend the small body of work on this industry standard.

Portability in cluster-based network servers is also achievable in ways that differ from our inter-node communication approach [7]. Two common approaches are to use: (1) content-oblivious dispatchers or front-ends; or (2) HTTP redirection. The first approach does not allow for considering cache locality or intelligently assigning requests to nodes, since the content requested is not inspected. The second approach allows for considering both locality and load balancing, but is not transparent to clients and increases response time significantly.

6 Conclusions

In this paper we have studied the efficiency vs. portability trade-off in cluster-based locality-conscious network servers using modeling and experimentation. Our modeling results assessed the potential benefits of portable and non-portable locality-conscious servers. Based on our experience with the model, we proposed and evaluated a portable, locality-conscious WWW server. Experiments with our server, a non-portable server, and a locality-oblivious server validated and confirmed our modeling results under several real workloads. We found our portable server to be efficient as well, given its performance relative to the other servers and how closely it approximates the upper bound on performance generated by our model.

Based on our modeling and experimental results, our main conclusion is that portability should be promoted in network servers implemented for fast-communication clusters, given its relatively small cost ($\leq$ 15%) in terms of performance. For slow-communication clusters, efficiency should be the overriding concern, as the performance cost of portability can be very high (as high as 98% on 32 cluster nodes). This conclusion is especially important to popular network service providers, which can avoid server re-implementations for each new generation of operating system or hardware platform at a small performance cost by using our server.

We also conclude that user-level communication can be be extremely useful, even for non-scientific applications such as network servers. This conclusion is especially important to system-area network protocol designers and system-area network manufacturers, since the previous work on user-level communication had not proven their case for these applications.

Acknowledgements

We would like to thank Liviu Iftode, Rich Martin, Thu Nguyen, and Tao Yang for discussions on the topic of this paper. We are also grateful to Evangelos Markatos for giving us the Forth trace.

References

[1] M. Arlitt and C. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, May 1996.

[2] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable Content-Aware Request Distribution in Cluster-Based Network Servers. In *Proceedings of USENIX'2000 Technical Conference*, June 2000.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: HyperText Transfer Protocol - HTTP/1.0. Technical report, HTTP Working Group, May 1996.

[4] A. Bestavros, M. Crovella, J. Liu, and D. Martin. Distributed Packet Rewriting and its Application to Scalable Server Architectures. In *Proceedings of the International Conference on Network Protocols*, October 1998.

[5] R. Bianchini and E. V. Carrera. Analytical and Experimental Evaluation of Cluster-Based WWW Servers. *World Wide Web journal*, 2000. Earlier version published as TR 718, Department of Computer Science, University of Rochester, August 1999, Revised April 2000.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of IEEE InfoCom'99*, pages 126–134, March 1999.

[7] V. Cardellini, M. Colajanni, and P. Yu. Dynamic Load Balancing on Web-Server Systems. *IEEE Internet Computing*, 3(3):28–39, May 1999.

[8] E. V. Carrera and R. Bianchini. Evaluating Cluster-Based Network Servers. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, pages 63–70, August 2000.

[9] F. Chong, R. Barua, F. Dahlgren, J. Kubiatowicz, and A. Agarwal. The Sensitivity of Communication Mechanisms to Bandwidth and Latency. In *Proceedings of the 4th International Symposium on High-Performance Computer Architecture*, January 1998.

[10] Cisco LocalDirector. http://www.cisco.com/, 2000.

[11] M. Dahlin, R. Yang, T. Anderson, and D. Paterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation*, November 1994.

[12] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A Scalable and Highly Available Web Server. In *Proceedings of COMPCON'96*, pages 85–92, 1996.

[13] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In *Proceedings of ACM SIGCOMM'98*, pages 254–265, 1998.

[14] 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.

[15] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. Cluster-Based Scalable Network Services. In *Proceedings of the International Symposium on Operating Systems Principles*, pages 78–91, 1997.

[16] D. Gross and C. M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc., 1998.

[17] V. Holmedahl, B. Smith, and T. Yang. Cooperative Caching of Dynamic Content on a Distributed Web Server. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, pages 243–250, July 1998.

[18] IBM SecureWay Network Dispatcher. http://www.software.ibm.com/network/dispatcher, 2000.

[19] R. Martin, A. Vahdat, D. Culler, and T. Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In *Proceedings of the 24th International Symposium on Computer Architecture*, June 1997.

[20] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, October 1998.

[21] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An Efficient and Portable Web Server. In *Proceedings of USENIX'99 Technical Conference*, June 1999.

[22] M. Rangarajan and L. Iftode. Software Distributed Shared Memory over Virtual Interface Architecture: Implementation and Performance. In *Proceedings of the 3rd Extreme Linux Workshop*, October 2000.

[23] Piranha – Load-Balanced Web and FTP Clusters. http://www.redhat.com/support/wpapers/piranha/, 2000.

[24] Resonate Central Dispatch. http://www.resonateinc.com/, 1999.

[25] Y. Saito, B. Bershad, and H. Levy. Manageability, Availability and Performance in Porcupine: A Highly Scalable, Cluster-Based Mail Service. In *Proceedings of 17th ACM Symposium on Operating Systems Principles*, December 1999.

[26] ServerIron. http://www.foundrynetworks.com/products/Webswitches.html, 2000.

[27] 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. In *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, January 2000.

[28] H. Zhu, B. Smith, and T. Yang. A Scheduling Optimization for Resource-Intensive Web Requests on Server Clusters. In *Proceedings of the 11th Annual Symposium on Parallel Algorithms and Architectures*, pages 13–22, June 1999.