Scalable Web Server Cluster Design with Workload-Aware Request Distribution Strategy WARD

Ludmila Cherkasova and Magnus Karlsson
Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94303
e-mail: {cherkasova,karlsson}@hpl.hp.com

Abstract. *In this work, we consider a web cluster in which the content-aware distribution is performed by each of the node in a web cluster. Each server in the cluster may forward a request to another node based on the requested content. We propose a new Workload-Aware Request Distribution strategy WARD, that assigns a small set of most frequent files, called core, to be served locally, by any server in a cluster, while partitioning the rest of the files to be served by different cluster nodes. We propose an algorithm, called ward-analysis, to compute the nearly optimal core size. The algorithm takes into account workload access patterns and cluster parameters such as number of nodes, node RAM, TCP handoff overhead, and disk access overhead.*

Our simulations driven by a realistic workload show that WARD achieves super-linear speedup with increased cluster size. It shows superior performance compared with traditional round-robin strategy (up to 260% increased throughput for a cluster of 16 nodes), and outperforms a pure partitioning strategy based on a cache-affinity requests distribution (up to 50% increased throughput for a cluster of 16 nodes).

1 Introduction and Background

Servers based on clusters of workstations are the most popular configuration used to meet the growing traffic demands imposed by the World Wide Web. A cluster of servers, arranged to act as a single unit, provides incremental scalability as it has the ability to grow gradually with demand. However, for clusters to be able to achieve the scalable performance with increase in cluster size, it is important to employ the mechanisms and policies for balanced request distribution.

Load balancing solutions can be represented by two major groups: 1) DNS based approaches; 2) IP/TCP/HTTP redirection based approaches. The second group, IP/TCP/HTTP redirection based approaches, employs a specialized *front-end node*, the load-balancer, which traditionally determines the least loaded server (this is the job of the proprietary algorithms implemented in different products) to which server in a cluster the packet has to be sent (see surveys [3, 8]).

Content-aware request distribution, on the other hand, takes into account the content (URL, URL type, or cookies) when making a decision to which back-end server the request has to be routed. Previous work on content-aware request distribution [2, 4, 7, 11] has shown that policies distributing the requests based on cache affinity lead to significant performance improvements compared to the strategies taking into account only load information.

To outline different cluster designs which were proposed to implement content-aware balancing strategies, we adopt terminology proposed in [2]. There are three main components comprising a cluster configuration with content aware request distribution strategy:

- the *dispatcher* which specifies which web server will be processing a given request;
- the *distributor* which interfaces the client and implements the mechanism that distributes the client requests to specific web server.
- *web server* which processes HTTP requests.

To be able to distribute the requests on base of requested content, the distributor component should implement some mechanism such as TCP handoff [7] or TCP splicing [6].

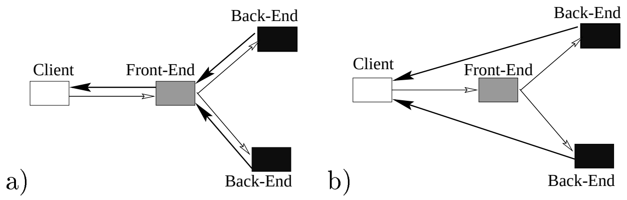

Figure 1: Traffic flow with a) Splicing mechanism, b) TCP handoff mechanism.

Splicing is an optimization of the front-end relaying approach, with the traffic flow represented in Figure 1 a). The TCP handoff was introduced in [7] to enable the forwarding of back-end responses directly to the clients as shown in Figure 1 b). This difference in the response flow route allows substantially higher scalability of the TCP handoff mechanism than TCP splicing. In [2], authors compared performance of both mechanisms showing the benefits of the TCP handoff schema.

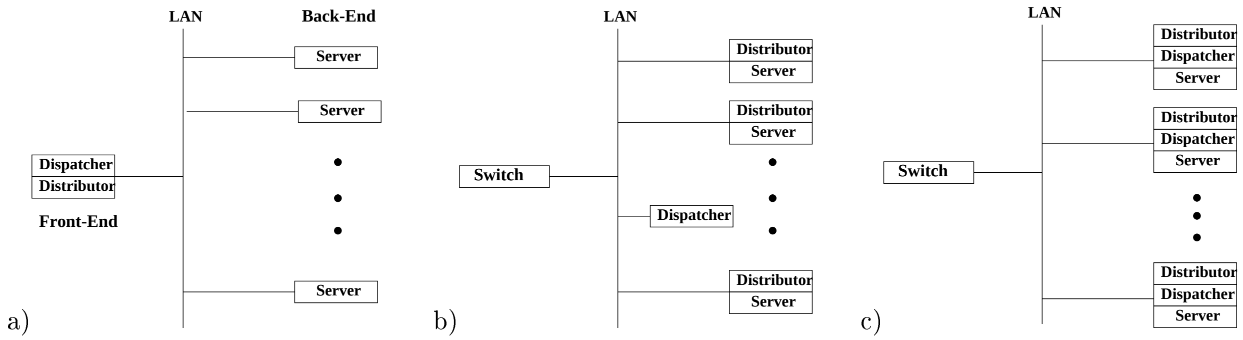

Figure 2: Different web server cluster architectures: a) with single front-end distributor, b) with co-located distributor and server, and a centralized dispatcher, c) with co-located distributor, server, and dispatcher.

For the rest of the paper, we assume that the distributor component implements some form of TCP handoff mechanism. Figure 2 a) shows the most typical cluster configuration with content aware request distribution strategy and a single front-end. In this configuration, the typical bottleneck is due to the front-end node which combines the functions of both the distributor and the dispatcher. For realistic workloads, the front-end node, performing the TCP handoff, does not scale well beyond four cluster nodes [2]. Most of the overhead in this scenario is incurred by the distributor component. Typically, only the dispatcher component requires centralized control.

Thus, another recent solution proposed in [2] is shown in Figure 2 b). It is based on alternative cluster design where the distributor components are co-located with the server components, while the dispatcher component is centralized. We will call this architecture CARD (Content-Aware Request Distribution). In this architecture the distributor is decoupled from the request distribution strategy defined by the centralized dispatcher module. The switch in front of the cluster can be a simple LAN switch or L4 level load-balancer. For simplicity, we assume that the clients directly contact a distributor, for instance via RR-DNS. In this case, the typical client request is processed if the following way. 1) Client web browser uses TCP/IP protocol to connect to the chosen distributor; 2) the distributor component accepts the connection and parses the request; 3) the distributor contacts the dispatcher for the assignment of the request to a server; 4) the distributor hands off the connection using TCP handoff protocol to the server chosen by the dispatcher (if it is not served locally) 5) the server sends the response directly to the client.

The results in [2] show good scalability properties of the proposed architecture above when distributing requests with the LARD policy [7]. The main idea behind LARD is to logically partition the documents among the cluster nodes, aiming to optimize the usage of the overall cluster RAM. Thus, the requests to the same document will be served by the same cluster node that will most likely have the file in RAM. However, under the described policy in a sixteen-node cluster, each node statistically will serve only 1/16 of the incoming requests locally and will forward 15/16 of the requests to the other nodes using the TCP handoff mechanism. TCP handoff is an expensive operation. That could lead to a significant *forwarding overhead*, decreasing the potential performance benefits of the proposed solution. From the other side, web server workload studies [1, 5] showed strong locality of references: 90% of the server requests target only 2-10% of all the accessed files, and constitute less than 5% of the total working set.

We propose a new request distribution strategy WARD (Workload-Aware Request Distribution) that takes workload properties into account and identifies a small set of most frequent files, called *core*. The requests to core files are processed by any server in a cluster, while the rest of the files are partitioned to be served by different cluster nodes. The goal of the WARD strategy is to minimize the forwarding overhead occurring from TCP handoff for the most frequent files while still optimizing the overall cluster RAM usage by partitioning the rest of the files that typically contribute to the largest portion of the working set. We have designed a special algorithm called *ward-analysis* that identifies the optimal size core for a given workload and given system parameters, such as number of nodes in a cluster, forwarding overhead, and *disk access overhead*. This way, WARD tunes itself to achieve the best performance trade-off for a given workload's access patterns and cluster configuration parameters.

Simulations driven by access logs from the HP.com web site show that WARD achieves super-linear speedup with increased cluster size. It shows excellent performance improvements compared to the traditional round-robin strategy by up to 260% in increased throughput for a 16-node cluster, and outperforms a partitioning strategy based on cache-affinity request distribution by up to 50% for a cluster of 16 nodes.

While the centralized dispatcher component of CARD simplifies the design and the distribution policy, it introduces the additional overhead and delay to send messages to the dispatcher and to get the responses with routing decisions back. The second question, we then address in this paper, is whether a centralized dispatcher component in CARD shown in Figure 2 b) can be distributed, or more exactly replicated across the nodes as shown in Figure 2 c), where the distributor and dispatcher components are co-located with the server components in what we call the *dec-CARD architecture*.

We propose a decentralized dispatcher architecture where the dispatcher components have the same routing information in all the nodes. In our approach, this routing information is updated on a daily basis, which means that *current day's* routing is performed with the routing created on the basis of *previous day's* access patterns. Our simulation results indicate that the performance of WARD based on the previous day's access log information is only 1%-15% worse in throughput than WARD using current day's workload information (via centralized dispatcher component).

The remainder of the paper presents our results in more detail. Section 2 formally introduces WARD strategy with emphasis on the *ward-analysis* algorithm. Section 3 presents our simulation results and their analysis, as well as a brief workload description and the simulation model.

2 A New Strategy WARD

It is well known that web server performance greatly depends on efficient memory usage. The throughput of a web server is higher when it reads pages from a cache in memory than from disk. If all files of the site fits in memory the web server demonstrates excellent performance because only the first request for a file will require a disk access, and all the following file accesses will be served from memory.

With WARD we attempt to achieve two goals:

1. maximize the number of requests served from the total cluster memory by partitioning files to be served by different servers;

2. minimize the *forwarding* by identifying the subset of *core* files to be processed on any node, i.e. allowing the replication of these files in the memories across the nodes. Note, that while serving the core files locally by each cluster node helps to minimize the forwarding overhead, it will most probably result in additional, initial disk accesses to core files on all those nodes. This is why the ultimate goal here is to identify such a subset of core files for which the forwarding overhead savings are higher than the additional cost of the disk accesses due to cold misses across the cluster nodes.

A cornerstone in our algorithm called *ward-analysis* is the use of *frequencies* (the number of times a file was accessed) and *sizes* of individual files. These are denoted $FileFreq$ and $FileSize$, respectively. We gather these by analyzing web-server access logs from the cluster. Let *Freq-Size* be the table of all accessed files with their frequency and the files sizes. This table is sorted in decreasing frequency order.

The algorithm presented in this section assumes that the cache replacement policy of the file cache in the web-server has the property that the most frequent files will most likely be in *cluster RAM*. [1] Here and for the rest of the paper we define *ClusterRAM* as the total size of all the file caches in the cluster.

If all the files were partitioned across the cluster the most probable files to be in the cluster RAM would be the most frequent files that fit into the cluster RAM. The starting point of our algorithm is the set of most frequent files that fit into the cluster RAM, called *BaseFiles* as shown in Figure 3 a). The other files we do not consider in the algorithm. Under the partition strategy PART, the maximum number of the *BaseFiles* are stored in the *ClusterRAM*, at a price that $\frac{N-1}{N}$ of the request coming to each node have to be handed off where N is the number of nodes in the cluster. Under the WARD strategy, *BaseFiles* are represented by three groups of files as shown in Figure 3 b): $Files_{core}$ and $Files_{part}$ in the *ClusterRAM*, and $Files_{on\ disk}$ consisting of files evicted from RAM to disk due to the expansion of the core. They satisfy the following equations:

$$BaseFiles = Files_{part} + Files_{core} + Files_{on\ disk} \quad (1)$$

$$N \cdot Size_{core} + Size_{part} \leq ClusterRAM \quad (2)$$

The *ideal case* for web server request processing is when

- a request is processed locally, i.e. it does not incur an additional *forwarding overhead* (denoted as $ForwardOH$); and

- a request is processed from the node RAM, i.e. it does not incur an additional *disk access* overhead (denoted as $DiskOH$).

Our goal is to identify a set of *core files* $Files_{core}$ that minimizes the total overhead due to *BaseFiles*:

$$OH_{BaseFiles} = OH_{part} + OH_{core} + OH_{on\ disk} \quad (3)$$

First, let us analyze what the additional overhead incurred by processing the requests to $Files_{part}$ is. Assuming all these files are partitioned to be served by different nodes, statistically a file in the partition incurs forwarding overhead on the average $(N-1)/N$ times. The file from partition will also incur one disk access on the node it is assigned to the first time it is read from disk. This reasoning gives us the following overhead for the partition files:

$$Penalty_{forward} = \frac{N-1}{N} \cdot FileFreq \cdot ForwardOH \quad (4)$$

[1] It is a reasonable assumption for the whole set of replacement policies.

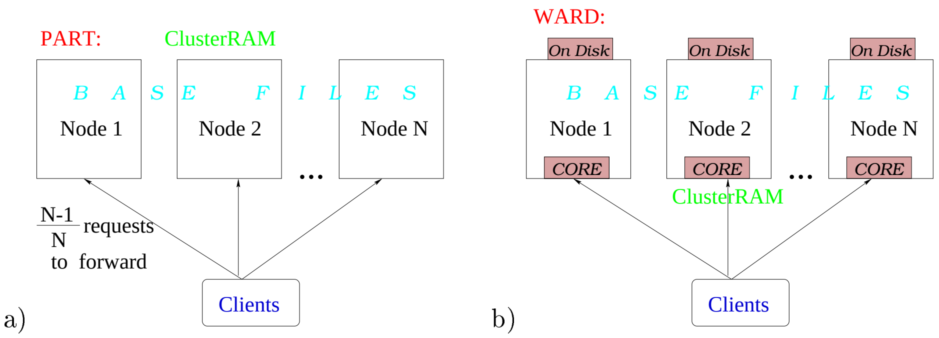

Figure 3: *BaseFiles* representation under: a) a partition strategy PART, b) a new strategy WARD.

$$Penalty_{DiskAccess} = FileSize \cdot DiskOH \quad (5)$$

$$OH_{part} = \sum_{Files_{part}} (Penalty_{forward} + Penalty_{DiskAccess}) \quad (6)$$

where $ForwardOH$ is the processing time in μsec the TCP handoff operation consumes, and $DiskOH$ is the extra time in μsec it generally takes to read one byte from disk compared to from RAM.

Now, let us understand what the additional overhead incurred by processing the requests to $Files_{core}$ is. If a file belongs to the core then the request to that file can be processed locally, i.e. with no additional forwarding overhead. The drawback is that the file has to be read from disk into memory once on all the nodes in the cluster, creating additional disk access overhead. However, this is under the assumption that the file is accessed frequently enough that at least one request for this file will end up on all nodes. For a file that is accessed less frequently this number is expected to be lower, thus we need to calculate the expected value of the number of nodes that get at least one access to a file given a certain frequency f and a number of nodes N.

$$E(f) = \sum_{i=1}^{N} i \cdot P(f, i) \quad (7)$$

Here $P(f, i)$ is the probability that exactly i nodes will have the file after f references to it. It can be calculate using the following recursion and starting conditions.

$$\begin{aligned} P(f+1, i) &= P(f, i-1) \cdot \frac{N-(i-1)}{N} + P(f, i) \cdot \frac{i}{N} \\ P(0,0) &= 1 \\ P(0,1) &= P(0,2) = \cdots = P(0,N) = 0 \\ P(1,0) &= P(2,0) = \cdots = P(\infty,0) = 0 \end{aligned}$$

The overhead due to extra disk accesses to core files can then be calculated as follows.

$$OH_{core} = \sum_{Files_{core}} E(FileFreq, N) \cdot DiskOH \cdot FileSize \quad (8)$$

Finally, the requests to $Files_{on\ disk}$ will incur additional disk overhead every time these files are accessed, which gives the following equation.

$$OH_{on\ disk} = \sum_{Files_{on\ disk}} FileFreq \cdot DiskOH \cdot FileSize \quad (9)$$

Using the reasoning and the equations above, a set $Files_{core}$ that minimizes the total overhead due to *BaseFiles* can be computed.

The next step is then to partition $Files_{part}$ across the cluster. We chose to partition the files in a balanced manner using load and file sizes, thus trying to put the same amount of data and load on all nodes.

The last step is to feed this information into each web-server node that acts on this in the following manner.

```
If in core:                    serve locally
If in partition and local:     serve locally
If in partition and remote:    hand off to
                               designated node
Everything else:               serve locally
```

3 Simulation Results

This Section presents simulation results for WARD strategy using realistic workload. This workload is briefly characterized in Section 3.1. We then describe the main parameters of our simulation model in Section 3.2. The simulation results and their analysis are reported in Section 3.3, which starts with a comparison of how well *ward-analysis* computes the core. After that we will compare WARD with round-robin and a pure partitioning strategy on both for the CARD architecture (Section 3.3) and the dec-CARD architecture (Section 3.4) .

3.1 Access Log

In our case study, we used access logs from the HP.com site (www.hp.com). HP.com provides diverse information about HP: business news, major events, detailed coverage of most software and hardware products, and press related news. For our simulations we used an access log consisting of 5 million that covers a few hours

Figure 4: Log Profile: a) density of references, b) working-set locality, c) bytes-transferred locality.

of daily traffic to the HP.com site. These 5 million entries contain only successful responses with code 200, which are responsible for all of the files transferred by the server. The absolute number of files in this access log was 69,067, and the total working set of the log was 2016.6 MB. Figure 4 shows a brief access log profile of the site. Figure 4a) reflects the reference locality: 90% of the server requests target only 4.5% of the files. In order to analyze the density of references against the working set size, we use a metric called *working-set locality*. It is defined as the percentage of the working set that the most frequently accessed files occupy, that contribute to a specific percentage of the total number of requests. Figure 4b) shows that 95.5% of all requests are to files that constitute only 5% of the total access log working set. Figure 4c) shows the bytes transferred due to these requests: 65% of all the bytes transferred by the site are due to these frequent files.

3.2 Simulation Model

Our simulation model was written using an event driven simulator called C++Sim [9]. The model makes the following assumptions about the capacity of each web server in the cluster:

- Web server throughput is 1000 req/sec when retrieving files of size 14.5 KB from the RAM (14.5 KB is the average file size under the SpecWeb96 benchmark [10]).
- Web server throughput is 10 times lower, i.e. 100 req/sec, when retrieving files from disk.[2]
- The file service time is proportional to the file size.
- The cache replacement policy is LRU.

In the simulations, we used two values for the forwarding overhead: 138 μsec and 276 μsec. The experimental results reported in [2] show that TCP handoff processing overhead is nearly 300 μsec just for the distributor part. This result influenced our choice of values for the forwarding overhead. Our goal is to understand the efficiency and performance benefits of WARD for current TCP handoff implementation, and to perform a sensitivity analysis on how these results change if the TCP handoff implementation imposes smaller overhead (in our case study, twice as small). From these results, we can interpolate a trend for the performance benefits if the TCP handoff imposes a smaller overhead.

In order to run the model, the requests from the original trace were split into N sub-traces, where N is the number of servers in a cluster (in round-robin manner, since we assume the simplest RR-DNS routing of the initial clients request to the nodes in the cluster). These sub-traces were then fed to the respective servers. Each server has a finite waiting queue for both request stemming from the RR-DNS routing and forwarded requests. The server grabs the next request from its sub-trace as soon as it is finished with the previous request. As the performance metrics, we have measured server throughput averaged across the servers after processing all the requests, and the cluster speedup achieved on the processing of the entire access log.

3.3 Main Simulation Results

Our first goal is to check whether our analysis identifies the core and partition sizes close to their optimal values. In order to do this, we ran a set of simulations with varied core sizes.

Figure 5 shows simulation results for four, eight, and sixteen-node clusters with a forwarding overhead of 276 μsec. The RAM size in each node was set to 5% of the access log's working set. This RAM size per node deserves some special attention, because for a sixteen-node cluster, the overall cluster RAM is $5\% \cdot 16 = 80\%$ of the total access log working set. In our simulation, for RAM sizes larger than 5% of the working set per node and 16-node cluster, WARD shows a little additional performance improvements, because most of the working set already fits in RAM of this large cluster.

In Figure 5, the line labeled WARD shows the average server throughput when varying the core size as a percentage of the RAM in the node. The left most point on this graph corresponds to the WARD strategy

[2] We measured web server throughput on an HP 9000/899 machine running HP-UX 11.00 when it supplied files from the file buffer cache, and compared it against the web server throughput when it supplied files from the disk. The difference in throughput was a factor of 10. For machines with different configurations, this factor will vary.

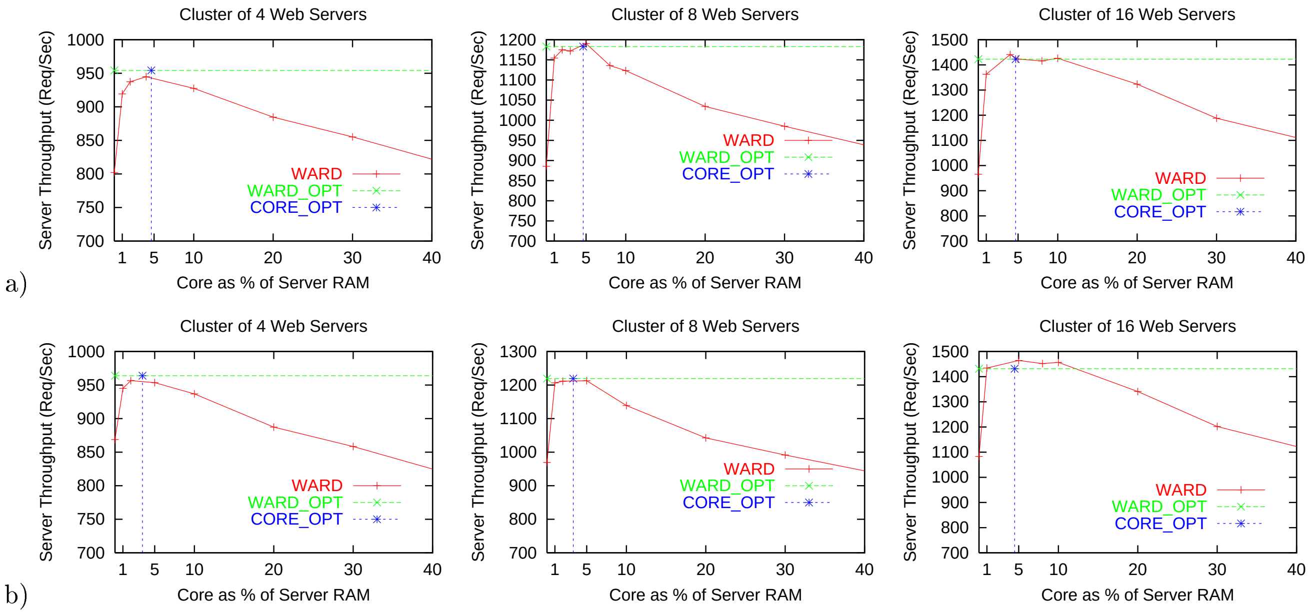

Figure 5: Graphs showing how well we compute optimal *core* and *partition* parameters for web cluster to achieve best performance for this workload. The RAM of each node is 5% of the access log working set, and the forwarding overhead is a) 276 μsec, b) 138 μsec.

with zero core size, i.e. the whole working set is partitioned among the servers. We will call this strategy PART. Intuitively, we expect that WARD with correctly chosen core and partition sizes will outperform PART.

Figure 5 shows that it is beneficial to replicate a small amount of most frequent files to avoid the forwarding overhead. However at some point, when less frequent documents are included in the core and thus replicated among the cluster nodes, the additional disk accesses occurring as a result of cold misses to these documents overweight the benefits gained from eliminated forwarding overhead for those files. The intersections of the other two lines correspond to the simulation results of WARD run with the core size computed as a results of the *ward-analysis* algorithm described in Section 2. It shows that indeed under computed core and partition sets, the web cluster performance is within 1% of the best performance for this workload. Thus good matches between the computed and simulated optimal core and partition sizes are observed for this workload.

Figure 5 b) shows similar simulation results for the clusters with a forwarding overhead of 138 μsec. This set of simulation results shows that a smaller forwarding overhead leads to a smaller size core. The intuitive explanation is that since the savings on the forwarding overhead against the price of the additional disk accesses are decreased, the files should reach a higher access frequency to justify their replication via the core mechanism.

Figure 6 shows the percentage of requests that can be served locally using WARD with varying sizes of the core. Additionally, there is a line that shows the percentage of requests that were served locally from the core, and a line representing the requests served locally under the strategy PART, where the core is zero and the whole working set is partitioned among the servers. Figure 6 shows that 80%-90% of the requests are served locally under WARD with optimal core, while under PART, the percentage of local requests is proportional to $1/N$ where N is the number of nodes. This figure demonstrates the intuition and intent that lays the foundation for WARD: **by mapping a small set of most frequent files to be served by multiple number of nodes, we can improve both locality of accesses and the cluster performance significantly**.

The next set of simulation results shown in Figure 7 shows the performance benefits of WARD compared to the Round-Robin request distribution and PART strategy. For a four-node cluster, we evaluate performance results for three different configurations: with RAM set to 5%, 10%, and 15% of the access log working set, resulting in 20%, 40%, and 60% of the entire working set covered by the combined cluster RAM. For a eight-node cluster, we use two different configurations: RAM set to 5% and 10% of the access log working set, corresponding to 40% and 80% of the entire working set covered by the combined cluster RAM. Finally, for the sixteen-node cluster, we set the RAM of the node to 5%, corresponding to 80% of entire working set fitting to the combined cluster RAM.

In Figure 7, WARD and PART have a forwarding overhead of 276 μsec and 138 μsec respectively. Under

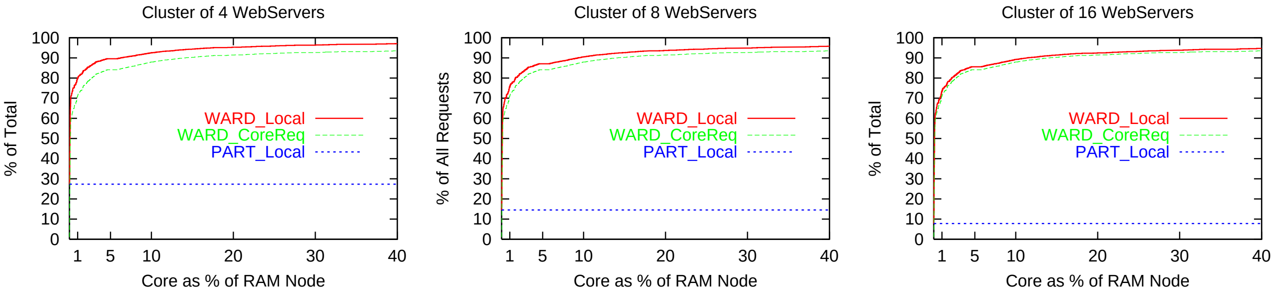

Figure 6: Percentage of local requests for WARD and PART strategies. The RAM of each node is 5% of the access log working set, and the forwarding overhead is 276 μsec.

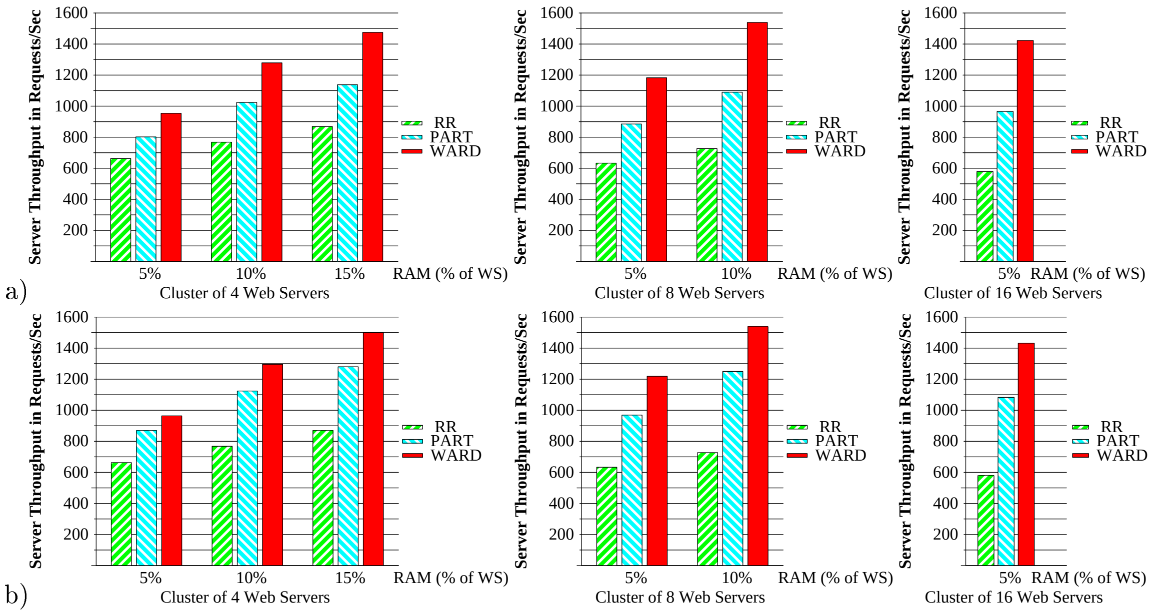

Figure 7: Average server throughput under Round-Robin (RR), PART, and WARD. The forwarding overhead is a) 276 μsec, b) 138 μsec.

the traditional Round-Robin (RR) strategy, servers are processing any incoming request locally, and do not require TCP handoff, and thus performance of RR does not depend on the forwarding overhead cost.

WARD shows significant improvements compared to RR and PART. The performance benefits of WARD increase with the number of nodes. In a sixteen-node cluster, RR suffers from redundant replication of files across all the nodes in the cluster, leading to inefficient RAM usage and additional disk accesses. PART, on the other hand, utilizes overall RAM efficiently by partitioning the files in the working set among the different nodes, but 15/16 of the traffic coming to each node has to be forwarded to some other node. The higher number of nodes, the more PART suffers from this forwarding overhead.

Figure 8 shows speed-up achieved under the strategies RR, PART, and WARD for clusters of different size, normalized to the performance of a one-node configuration.

For the case when the forwarding overhead is 276 μsec, the sixteen-node configuration with PART, for example, shows a speed-up of 21, and the sixteen-node configuration with WARD shows a speed-up of 31. RR for a sixteen-node cluster achieves only a speed-up of 12, showing less than perfect speed-up for large cluster sizes. In summary, Figure 8 a) shows that 1) WARD outperforms RR by up to 2.6 times, and 2) WARD demonstrates up to 50% performance improvement over PART. Figure 8 b) shows similar results for the cluster configuration with a forwarding overhead of 138 μsec. As expected, the difference between the performance of PART and WARD gets smaller, but for a sixteen-node cluster, we still see up to 32% of improvement in throughput. WARD compared with traditional RR shows up to 2.5 times improvement in performance.

Figure 8 shows that for eight- and sixteen-node clusters, both PART and WARD achieve super-liner speed-ups. The logic behind it is the following. A cluster with

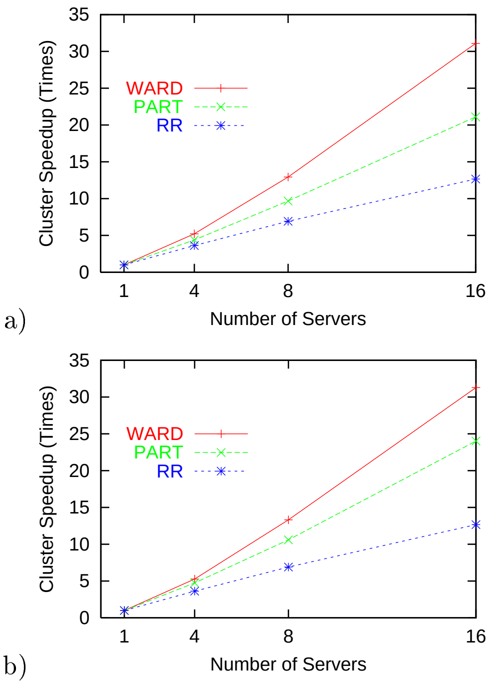

Figure 8: Cluster speed-up under different strategies. The RAM of each node is 5% of the access log's working set, and the forwarding overhead is a) 276 μsec, b) 138 μsec.

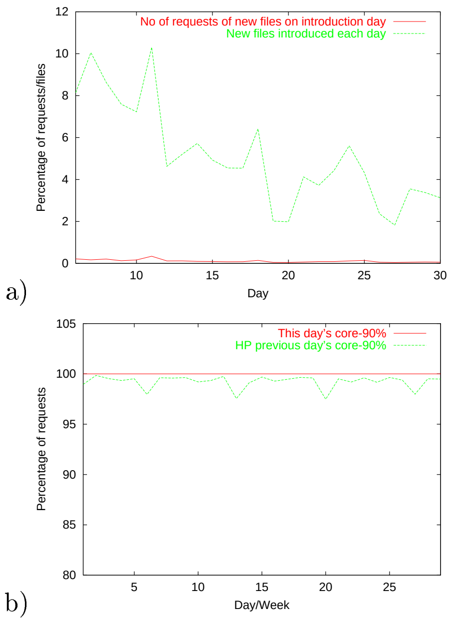

Figure 9: a) The percentage of new files introduced each day relative to the number of files accessed that day, and the percentage of requests to the new files on the introduction day, b) the relative number of requests that are to previous core files each day.

sixteen nodes ideally should process the same trace about sixteen times faster than a single node, due to sixteen times more processors. However, a cluster with sixteen nodes also has sixteen times more memory. The speedup under Round-Robin strategy is mostly due to increased processing power. PART and WARD take advantage of both the increased memory and increased processing power and thus demonstrate a super-linear speed-up as the cluster size increases.

3.4 Simulation Results for *dec*-CARD architecture

The next question we would like to address is whether the centralized dispatcher can be decentralized, as shown in Figure 2 c), and use a precomputed mapping or routing based on the access log information gathered from the previous day. [3] This idea makes sense, if a web server has a predictable and stable enough access pattern to its most frequent files.

As WARD on dec-CARD is based on previous day's access pattern it will not have information about accesses to new files introduced on the current day, and will serve these files locally. The first natural step is then to observe the introduction of new files in the logs, and to analyze the portion of all requests for those files. If there are few requests to newly introduced files the performance impact of them will be low. Interestingly, even if there are new files with relatively high number of accesses, by serving them locally it will be equivalent to mapping them to a core, and this gives us a performance benefit. For the rarely accessed files the chance is high that they are on disk anyway.

We analyzed how HP.com's stability and access traffic pattern change on a daily basis with our **WebMetrix** tool [5]. Figure 9 a) shows two curves: the percentage of new files introduced each day relative to the number of files accessed that day, and the percentage of requests to these files on the introductory day relative to the total number of request that day. A file is considered new if the file name has not been encountered before during the measurement interval. There is no statistics for the first week as it is used as a warm-up period. This diagram shows that even if the relative amount of new files introduced on a certain day can be high, the percentage of the requests due to these new files on the introduction day is low. So for HP.com, the performance will not be degraded by much.

The next thing we ask ourselves is how much of the core that changes between days as this affects the stability of the core. If the core is unstable our approach of performing today's routing with yesterday's routing information will generally not perform well. A key concept in this discussion is the locality of reference. Web server workloads generally exhibit high locality

[3] The other way to think about this is not necessarily on a "daily" basis. The routing in this replicated dispatchers can be updated more often, e.g. every 15 minutes or so. Our intent was to see how much inefficiency this "outdated" routing can cause.

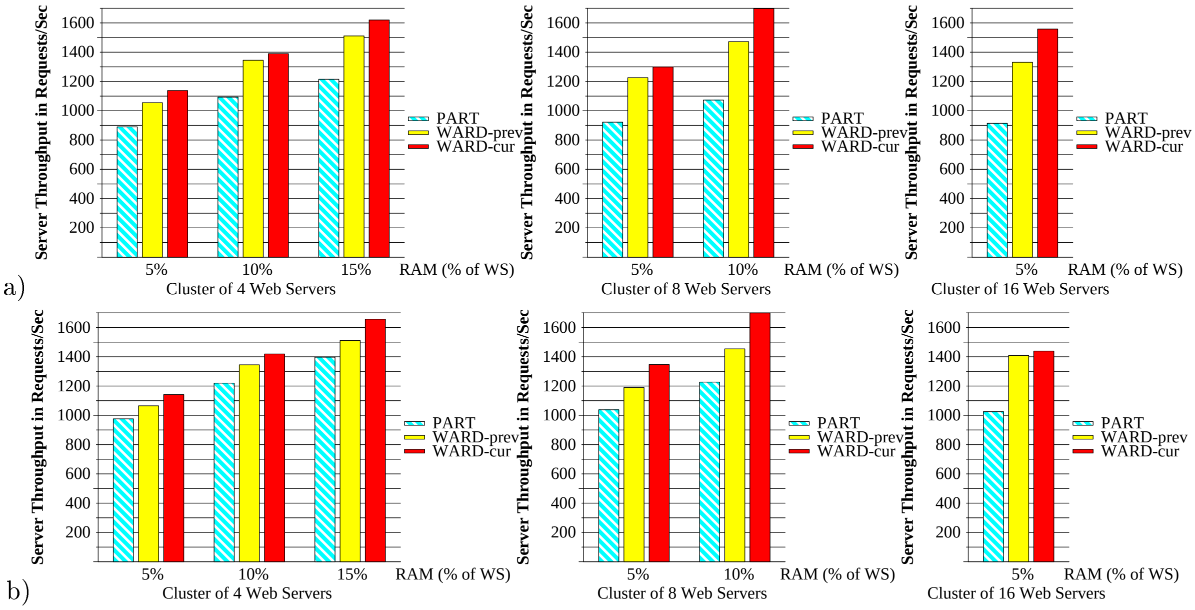

Figure 10: Average server throughput for PART based on current day's access patterns, WARD with current day's access pattern and previous day's access pattern. The forwarding overhead is a)276 μsec, b) 138 μsec.

of references. Figure 4 from Section 3.1 shows that 90% of the server requests come to only 4.5% of the files. Thus, this small set of files has strong impact on the web server performance. We define the *core-90%* as the set of most frequently accessed files that makes up for 90% of the requests. Going back to Figure 4, 90% of the requests to HP.com constitutes less than 2% of its working sets. From a performance point of view it is these core-90% files we should concentrate on to obtain good performance as most accesses are to them. Figure 9 depicts how the core-90% changes over days. More specifically, it shows the relative number of requests that are to previous core-90% files each day. From the figure we can see that less than 3% of the requests are due to accesses to new core-90% files. Thus the most performance critical working set is quite steady over time.

Our analysis of the access patterns for HP.com shows predictable and stable traffic patterns to the most frequent and popular documents on the site. For such stable sites, the cluster design shown in Figure 2 c), with WARD request distribution policy based on the previous day's information, works well, minimizing the overhead due to dispatcher and distributor components.

Note, that in the following simulation results, we did not include any performance benefits from having the distributors replicated across all nodes instead of a single centralized one. The results only reflect how well WARD performs when it is using the previous day's access log information.

Figure 10 a) shows the average server throughput achieved under WARD based on previous day's access log information, compared to WARD and PART based on the current day's access patterns. The forwarding overhead is 276 μsec. As seen from the graph, WARD based on the previous day's access log information shows good performance results. It is only 1%-13% worse than the WARD strategy using current day's workload information. Figure 10 b) shows similar results for a cluster with a forwarding overhead of 138 μsec. The performance of WARD based on the previous day's access log is only 1%-15% worse than WARD using current day's workload information.

Taking into account the additional performance benefits due to distributed dispatcher architecture that were not included in the above simulation results, we conclude that the cluster architecture shown in Figure 2 c) with WARD based on access log information from the previous day is an interesting solution for a scalable web server cluster.

4 Conclusion

Research in scalable web server clusters has received much attention from both industry and academia. A routing mechanism for distributing requests to individual servers in a cluster is at the heart of any server clustering technique.

Web server scalability and performance greatly depends on efficient RAM usage. The throughput of a web server is much higher when the requested documents are found in a server RAM. Previous work on content-aware request distribution (LARD, HACC, and FLEX strategies) [2, 7, 11, 4] has shown that policies distributing the requests based on cache affinity lead to significant performance improvements compared to the strategies taking into account only the load information. Content-aware request distribution strategies produce super-linear speed-ups with increasing cluster size. This is possible due to efficient utilization of both additional processing power and additional RAM of the new nodes.

In this work, we consider a web cluster design proposed in [2] for content-aware request distribution. The experimental results from [2] show that TCP handoff is quite an expensive operation. Besides, the cost of the TCP handoff mechanism can vary depending on the implementation and specifics of the underlying hardware. It could lead to significant *forwarding overhead*, decreasing the potential performance benefits of the proposed solution in [2].

We propose a new Workload-Aware Request Distribution strategy called WARD, which assigns a small set of most frequent files, called *core*, to be served locally, by any server in a cluster, while partitioning the rest of the files that fit into the combined cluster RAM to be served by different cluster nodes. We propose an algorithm, called *ward-analysis*, to compute the core size by taking into account workload access patterns and the cluster parameters such as number of nodes, node RAM, TCP handoff overhead, and disk access overhead.

WARD shows excellent performance improvements compared to traditional round-robin strategy by up to 260% increased throughput for a cluster of 16 nodes, and outperforms a partitioning strategy based on a cache-affinity requests distribution by up to 50% increased throughput for the same cluster size. Our sensitivity analysis shows that the benefits of WARD are higher for larger size clusters, as well as when the overhead due to TCP handoff implementation is higher.

Additionally, we investigate the idea of distributed, replicated dispatchers that use previous day's access patterns information to compute the routing information for current day. Simulation results for HP.com show that this cluster architecture is an interesting offering that further improves the scalability properties of web cluster solutions for sites with stable and predictable access patterns.

Our future efforts concentrate on building a prototype to verify the simulation results reported in this paper, as well as further, quantitative refinement of the metrics to navigate between design and policy choices.

5 Acknowledgments

We would like to thank Dick Carter for his advice on computing probabilities in formula 7 in Section 2. We also appreciate useful critique from our colleagues Wenting Tang and Lance Russell with whom we are currently working on a WARD prototype. Finally, we would like to thank the people from the ScalaServer project at Rice University: Mohit Aron, Vivek Pai, Peter Drushel and Willy Zwaenepoel for sharing their TCP handoff code, and answering our questions related to it.

References

[1] M. Arlitt and C. Williamson. Web server workload characterization: the search for invariants. In *Proceedings of the ACM SIGMETRICS Conference*, pages 126–137, 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 the USENIX 2000 Annual Technical Conference.*

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

[4] L. Cherkasova. FLEX: Load balancing and management strategy for scalable web hosting service. In *Proceedings of the Fifth International Symposium on Computers and Communications (ISCC'00)*, pages 8–13, July 2000.

[5] L. Cherkasova and M. Karlsson. Dynamics and evolution of web sites: Analysis, metrics and design issues. In *To be published at IEEE International Symposium on Computer Communications and Networks(ISCC'01)*, July 2001.

[6] A. Cohen, S. Rangarajan, and H. Slye. On the performance of tcp splicing for url-aware redirection. In *Proceedings of the 2nd Usenix Symposium on Internet technologies and Systems.*

[7] V. Pai, M. Aron, M. Svendsen, P. Drushel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, pages 205–216, October 1998.

[8] T. Schroeder, S. Goddard, and B. Ramamurthy. Scalable web server clustering technologies. *IEEE Network*, 14(3):38–45, 2000.

[9] H. Schwetman. Object-oriented simulation modeling with c++/csim. In *Proceedings of 1995 Winter Simulation Conference*, pages 529–533, December 1995.

[10] *The Workload for the SPECweb96 Benchmark.* http://www.specbench.org/osg/web96/workload.html.

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