"A Study on Locality-Aware Request Distribution in Cluster-Based Network Servers"

Anuradha

Assistant Professor, Dronacharya Institute of Management & Technology, Kurukshetra, Haryana - 136118

Abstract – We consider cluster-based network servers in which a front-end directs inward requests to one of a number of back-ends. Specifically, we consider content-based request distribution: the front-end uses the content requested, in addition to information about the load on the back-end nodes, to choose which back-end will handle this request. Content-based request distribution can improve locality in the back-ends' main memory caches, increase secondary storage scalability by partitioning the server's database, and provide the ability to employ back-end nodes that are specialized for certain types of requests.

----------♦----------

INTRODUCTION

Network servers based on clusters of commodity workstations or PCs connected by high-speed LANs combine cutting-edge performance and low cost. A cluster-based network server consists of a front-end, responsible for request distribution and a number of back-end nodes, responsible for request processing. The use of a front-end makes the distributed nature of the server transparent to the clients. In most current cluster servers the frontend distributes requests to back-end nodes without regard to the type of service or the content requested.

That is, all back-end nodes are considered equally capable of serving a given request and the only factor guiding the request distribution is the current load of the backend nodes.

With content-based request distribution, the frontend takes into account both the service/content requested and the current load on the back-end nodes when deciding which back-end node should serve a given request.

The potential advantages of content-based request distribution are:

(1) Increased performance due to improved hit rates in the back-end's main memory caches,

(2) Increased secondary storage scalability due to the ability to partition the server's database over the different back-end nodes and

(3) The ability to employ back-end nodes that are specialized for certain types of requests (e.g., audio and video).

The locality-aware request distribution (LARD) strategy presented in this paper is a form of content-based request distribution, focusing on obtaining the first of the advantages cited above, namely improved cache hit rates in the back-ends. Secondary storage scalability and special-purpose back-end nodes are not discussed any further in this paper.

Figure 1 illustrates the principle of LARD in a simple server with two back-ends and three targets' (A, B, C) in the incoming request stream. The front-end directs all requests for A to back-end 1, and all requests for B and C to back-end 2. By doing so, there is an increased likelihood that the request finds the requested target in the cache at the back-end. In contrast, with a round-robin distribution of incoming requests, requests of all three

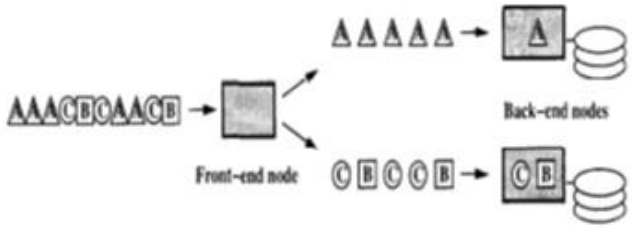


Figure 1: Locality-Aware Request Distribution

targets will arrive at both back-ends. This increases the likelihood of a cache miss, if the sum of the sizes of the

three targets, or, more generally, if the size of the working set exceeds the size of the main memory cache at an individual back-end node.

Of course, by naively distributing incoming requests in a content-based manner as suggested in Figure 1, the load between different back-ends might become unbalanced, resulting in worse performance. The first major challenge in building a LARD cluster is therefore to design a practical and efficient strategy that simultaneously achieves load balancing and high cache hit rates on the back-ends. The second challenge stems from the need for a protocol that allows the front-end to hand off an established client connection to a back-end node, in a manner that is transparent to clients and is efficient enough not to render the front-end a bottleneck. This requirement results from the front-end's need to inspect the target content of a request prior to assigning the request to a back-end node. This paper demonstrates that these challenges can be met, and that LARD produces substantially higher throughput than the state-of the-art approaches where request distribution is solely based on load balancing, for workloads whose working set exceeds the size of the individual node caches.

Increasing a server's cache effectiveness is an important step towards meeting the demands placed on current and future network servers. Being able to cache the working set is critical to achieving high throughput, as a state-of-the-art disk device can deliver no more than 120 block requests/set, while high-end network servers will be expected to serve thousands of document requests per second.

Moreover, typical working set sizes of web servers can be expected to grow over time, for two reasons. First, the amount of content made available by a single organization is typically growing over time. Second, there is a trend towards centralization of web servers within organizations. Issues such as cost and ease of administration, availability, security and high-capacity backbone network access cause organizations to move towards large, centralized network servers that handle all of the organization's web presence. Such servers have to handle the combined working sets of all the servers they supersede.

With round-robin distribution, a cluster does not scale well to larger working sets, as each node's main memory cache has to fit the entire working set. With LARD, the effective cache size approaches the sum of the node cache sizes. Thus, adding nodes to a cluster can accommodate both increased traffic (due to additional CPU power) and larger working sets (due to the increased effective cache size).

OBJECTIVES OF THE STUDY

The objectives of the current study are as follows:

1. To study practical and efficient LARD strategy that achieves high cache hit rates and good load balancing,

2. To study an efficient TCP protocol, that enables content-based request distribution by providing client transparent connection handoff for TCP-based network services and

3. To study a performance evaluation of a prototype LARD server cluster, incorporating the TCP handoff protocol and the LARD strategy.

NEED OF THE STUDY

As a specific policy for content-based request distribution, we introduce a simple, practical strategy for locality-aware request distribution (LARD). With LARD, the front-end distributes incoming requests in a manner that achieves high locality in the back-ends' main memory caches as well as load balancing. Locality is increased by dynamically subdividing the server's working set over the back-ends. Trace-based simulation results and measurements on a prototype implementation demonstrate substantial performance improvements over state-of-the-art approaches that use only load information to distribute requests. On workloads with working sets that do not fit in a single server node's main memory cache, the achieved throughput exceeds that of the state-of-the-art approach by a factor of two to four.

With content-based distribution, incoming requests must be handed off to a back-end in a manner transparent to the client, after the front-end has inspected the content of the request. To this end, we introduce an efficient TCP hand of protocol that can hand off an established TCP connection in a client-transparent manner.

HYPOTHESIS OF THE STUDY

The following hypothesis holds for a request distribution strategies considered in this paper:

1. The front-end is responsible for handing off new connections and passing incoming data from the client to the back-end nodes. As a result, it must keep track of open and closed connections, and it can use this information in making load balancing decisions. The frontend is not involved in handling outgoing data, which is sent directly from the back-ends to the clients.

2. The front-end limits the number of outstanding requests at the back-ends. This approach allows the frontend more flexibility in responding to changing load on the back-ends, since waiting requests can be directed to back-ends as capacity becomes available. In contrast, if we queued requests only on the back-end nodes, a slow node could cause many requests to be delayed even though other nodes might have free capacity.

REVIEW OF RELATED LITERATURE

Much current research addresses the scalability problems posed by the Web. The work includes cooperative caching proxies inside the network, push-based document distribution and other innovative techniques. Our proposal addresses the complementary issue of providing support for cost-effective, scalable network servers.

Robinson (2011) studied network servers based on clusters of workstations are starting to be widely used. Several products are available or have been announced for use as frontend nodes in such cluster servers. To the best of our knowledge, the request distribution strategies used in the cluster front-ends are all variations of weighted round-robin, and do not take into account a request's target content. An exception is the Dispatch product by Resonate, Inc., which supports content-based request distribution. The product does not appear to use any dynamic distribution policies based on content and no attempt is made to achieve cache aggregation via content-based request distribution.

Hunt et al. 2009 proposed a TCP option designed to enable content-based load distribution in a cluster server. The design has not been implemented and the performance potential of content-based distribution has not been evaluated as part of that work. Also, no policies for content-based load distribution were proposed.

Our TCP handoff protocol design was informed by Hunt et al.'s design, but chooses the different approach of layering a separate handoff protocol on top of TCP.

Fox et al. 2010 report on the cluster server technology used in the Inktomi search engine. The work focuses on the reliability and scalability aspects of the system and is complementary to our work. The request distribution policy used in their systems is based on weighted round Round-robin.

Loosely-coupled distributed servers are widely deployed on the Internet. Such servers use various techniques for load balancing including DNS round-robin,

Carter (2010) observed that HTTP client re-direction, Smart clients, source based forwarding and hardware translation of network addresses have problems related to the quality of the load balance achieved and the increased request latency.

Damani et al. 2011 IBM's Lava project loses the concept, of a "hit server". The hit server is a specially configured server node responsible for serving cached content. Its specialized OS and client-server protocols give it superior performance for handling HTTP requests of cached documents, but limits it to private intranets. Requests for uncached documents and dynamic content are delegated to a separate, conventional HTTP server node.

Johnson (2008) shares some of the the goals, but maintains standard client-server protocols, maintains support for dynamic content generation and focuses on cluster servers.

The LARD strategies result in a good combination of load balancing and locality. In addition, the strategies outlined above have while (true)

fetch next request r;

if serverSet[r.target] = 0 then

n, serverSet[r.target] t {least loaded node};

else

n t {least loaded node in serverSet[r.target]};

m t {most loaded node in serverSet[r.target]};

if (n.load > Thtgh && 3 node with load < Ti,,) 11

nload > 2 Thrgh then

p t {least loaded node};

add p to serverSet[r.target];

n + P;

if]serverSet[r.target]] > 1 &&

time0 - serverSet[r.target].lastMod > K then

remove m from serverSet[r.target];

send r to n

if serverSet[r.target] changed in this iteration then

serverSet[r.target].lastMod t time();

several desirable features.

First, they do not require any extra communication between the front-end and the back-ends. Second, the front-end need not keep track of any frequency of access information or try to model the contents of the caches of the back-ends. In particular, the strategy is independent of the local replacement policy used by the back-ends. Third, the absence of elaborate state in the front-end makes it rather straightforward to recover from a back-end node failure. The front-end simply re-assigns targets assigned to the failed back-end as if they had not been assigned before. For all these reasons, we argue that the proposed strategy can be implemented without undue complexity.

In a simple implementation of the two strategies, the size of the server or server-set arrays, respectively, can grow to the number of targets in the server's database.

Despite the low storage overhead per target, this can be of concern in servers with very large databases. In this case, the mappings can be maintained in an LRU cache, where assignments for targets that have not been accessed recently are discarded. Discarding mappings for such targets is of little consequence, as these targets have most likely been evicted from the back-end nodes' caches anyway.

RESEARCH METHODOLOGY

The input to the simulator is a stream of tokenized target requests, where each token represents a unique target being served. Associated with each token is a target size in bytes. This tokenized stream can be synthetically created, or it can be generated by processing logs from existing web servers.

The individual processing steps for a given request must be performed in sequence, but the CPU and disk times for differing requests can be overlapped. Also, large file reads are blocked, such that the data transmission immediately follows the disk read for each block.

Multiple requests waiting on the same file from disk can be satisfied with only one disk read, since all the requests can access the data once it is cached in memory.

The costs for the basic request processing steps used in our simulations were derived by performing measurements on a 300 Mhz Pentium 11 machine running FreeBSD 2.2.5 and an aggressive experimental web server. Connection establishment and teardown costs are set at 145~s of CPU time each, while transmit processing incurs 40~s per 512 bytes. Using these numbers, an 8 KB document can be served from the main memory cache at a rate of approximately 1075 requests/set.

If disk access is required, reading a file from disk has a latency of 28 ms (2 seeks + rotational latency). The disk transfer time is 410~s per 4 KB (resulting in approximately 10 MB/set peak transfer rate). For hales larger than 44 KB, an additional 14 ms (seek plus rotational latency) is charged for every 44 KB of file length in excess of 44 K. 44 KB was measured as the average disk transfer size between seeks in our experimental server. Unless otherwise stated, each back-end node has one disk.

Figures 2 and 3 show the cumulative distributions of request frequency and size for the Rice University trace and the IBM trace, respectively. Shown on the x-axis is the set of target files in the trace, sorted in decreasing order of request frequency. The y-axis shows the cumulative fraction of requests and target sizes, normalized to the total number of requests and total data set size, respectively. The data set for the Rice University trace consist of 37703 targets covering 1418 MB of space, whereas the IBM trace consists of 38527 targets and 1029 MB of space. While the data sets in both traces are of a comparable size, it is evident from the graphs that the Rice trace has much less locality than the IBM trace. In the Rice trace, 560/705/927 MB of memory is needed to cover 99% of all requests, respectively, while only 51/80/182 MB are needed to cover the same fractions of requests in the IBM trace.

This difference is likely to be caused in part by the different time spans that each trace covers. Also, the IBM trace is from a single high-traffic server, where the content designers have likely spent effort to minimize the sizes of high frequency documents in the interest of performance. The Rice trace, on the other hand, was merged from the logs of several departmental servers.

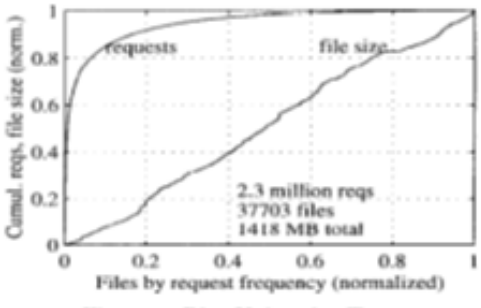
The cache replacement policy we chose for all simulations is Greedy-Dual-Size (GDS), as it appears to be the best known policy for Web workloads. We have also performed simulations with LRU, where files with a size of more than 500KB are never cached. The relative performance of the various distribution strategies remained largely unaffected. However, the absolute throughput results were up to 30% lower with LRU than with GDS.

As with all caching studies, interesting effects can only be observed if the size of the working set exceeds that of the cache. Since even our larger trace has a relatively small data set (and thus a small working set) and also to anticipate future trends in working set sizes, we chose to set the default node cache size in our simulations to 32 MB. Since in reality, the cache has to share main memory with OS kernel and server applications, this typically

requires at least 64 MB of memory in an actual server node.

SIMULATION INPUTS 3.3 SIMULATION OUTPUTS

The input to the simulator is a stream of tokenized target requests, where each token represents a unique target. The simulator calculates overall throughput, hit rate, and underutilization time.

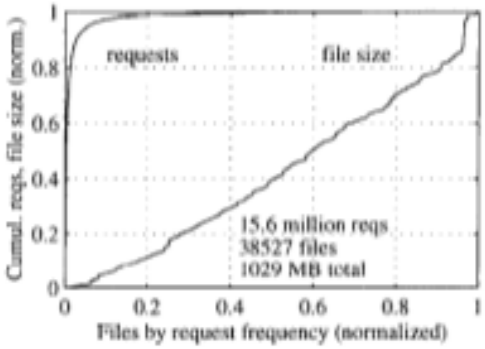


Throughput is the number of request in the trace that were served per second by the entire cluster, calculated as the number of requests in the trace divided by the simulated time it took to finish serving all the requests in the trace. The request arrival rate was matched to the aggregate throughput of the server.

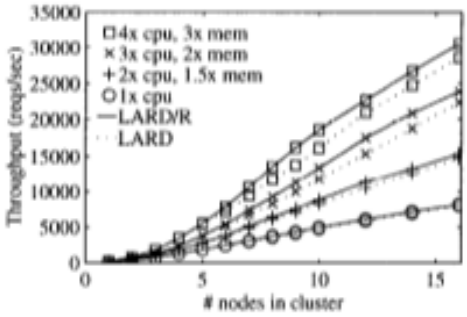
The cache hit ratio is the number of requests that hit in a back-end node's main memory cache divided by the number of requests in the trace. The idle time was measured as the fraction of simulated time during which a back-end node was underutilized, averaged over all back-end nodes.

In addition, observing the large amount of interest generated by global memory systems (GMS) and cooperative caching to improve hit rates in cluster main memory caches, we simulate a weighted round-robin strategy in the presence of a global memory system on the back-end nodes.

We simulate an idealized locality-based strategy, termed LB/GC, where the front-end keeps track of each back-end's cache state to achieve the effect of a global cache. On a cache hit, the front-end sends the requests to the back-end that caches the target. On a miss, the front-end sends the request to the back-end that caches the globally "oldest" target, thus causing eviction of that target.



LARD/R over LARD increases with CPU speed, even on a workload that presents little opportunity for replication.



To study various request distribution policies for a range of cluster sizes under different assumptions for CPU speed, amount of memory, number of disks and other parameters, we developed a configurable web server cluster simulator. We also implemented a prototype of a LARD-based cluster.

The simulation model is depicted in Figure 4. Each back-end node consists of a CPU and locally-attached disk(s), with separate queues for each. In addition, each node maintains its own main memory cache of configurable size and replacement policy. For simplicity, caching is performed on a whole-file basis.

Processing a request requires the following steps:

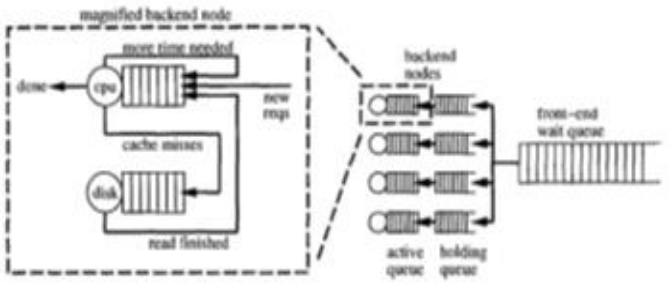


Figure 4: Cluster Simulation Model

connection establishment, disk reads (if needed), target data transmission, and connection teardown. The assumption is that front-end and networks are fast enough not to limit the cluster's performance, thus fully exposing the throughput limits of the back-ends. Therefore, the front-end is assumed to have no overhead and all networks have infinite capacity in the simulations.

Associated with each token is a target size in bytes. This tokenized stream can be synthetically created, or it can be generated by processing logs from existing web servers.

The individual processing steps for a given request must be performed in sequence, but the CPU and disk times for differing requests can be overlapped. Also, large file reads are blocked, such that the data transmission immediately follows the disk read for each block.

Multiple requests waiting on the same file from disk can be satisfied with only one disk read, since all the requests can access the data once it is cached in memory.

The costs for the basic request processing steps used in our simulations were derived by performing measurements on a 300 Mhz Pentium 11 machine running FreeBSD 2.2.5 and an aggressive experimental web server. Connection establishment and teardown costs are set at 145~s of CPU time each, while transmit processing incurs 40~s per 512 bytes. Using these numbers, an 8 KB document can be served from the main memory cache at a rate of approximately 1075 requests/set.

If disk access is required, reading a file from disk has a latency of 28 ms (2 seeks + rotational latency). The disk transfer time is 410~s per 4 KB (resulting in approximately 10 MB/set peak transfer rate). For hales larger than 44 KB, an additional 14 ms (seek plus rotational latency) is charged for every 44 KB of file length in excess of 44 KB. 44 KB was measured as the average disk transfer size between seeks in our experimental server. Unless otherwise stated, each back-end node has one disk.

CONCLUSION

We present and evaluate a practical and efficient locality-aware request distribution (LARD) strategy that achieves high cache hit rates and good load balancing in a cluster server. Trace-driven simulations show that the performance of our strategy exceeds that of the state-of-the-art weighted round-robin (WRR) strategy substantially. On workloads with a working set that does not fit in a single server node's main memory cache ,the achieved throughput exceeds that of WRR by a factor of two to four.

Additional simulations show that the performance advantages of LARD over WRR increase with the disparity between CPU and disk speeds. Also, our results indicate that the performance of a hypothetical cluster with WRR distribution and a global memory system (GMS) falls short of LARD under all workloads considered, despite generous assumptions about the performance of a GMS system.

We also propose and evaluate an efficient TCP handoff protocol that enables LARD and other content based request distribution strategies by providing client transparent connection handoff for TCP-based network services, like HTTP. Performance results indicate that in our prototype cluster environment and on our workloads, a single CPU front-end can support 10 back-end nodes with equal CPU speed as the front-end. Moreover, the design of the handoff protocols is expected to yield scalable performance on SMP-based front-ends, thus supporting larger clusters.

Finally, we present performance results from a prototype LARD server cluster that incorporates the TCP handoff protocol and the LARD strategy. The measured results confirm the simulation results with respect to the relative performance of LARD and WRR.

In this paper, we have focused on studying HTTP servers that serve static content. However, caching also be effective for dynamically generated content.

Moreover, resources required for dynamic content generation like server processes, executables, and primary data files are also cacheable. While further research is required, we expect that increased locality can benefit dynamic content serving, and that therefore the advantages of LARD also apply to dynamic content.

REFERENCES

- Carter et al. 2010 SWEB: Towards a Scalable WWW Server on Multi-Computers. In Proceedings of the 10th International Parallel Processing Symposium, Apr. 2010.
- Apache. http://www.apache.org/.
- Damini (2011). Optimistic Deltas for WWW Latency Reduction. In Proceedings of the 2011 Usenix Technical Conference, Jan. 2011.
- Hunt (2009). DNS Support for Load Balancing. RFC 1794, Apr. 2009.
- Fox(2010) Cost-aware WWW proxy caching algorithms. In Proceedings of the USENIX Symposium on Internet Technologies and Systems

(USITS), Monterey, CA, Dec. 1997.

- Johnson (2008). A Hierarchical Internet Object Cache. In Proceedings of the 1996 Use nix Technical Conference, Jan. 1996.
- Robinson (2011). Cooperative caching: Using remote client memory to improve file system performance. In Proc. Symp. on Operating Systems Design and Implementation, Monterey, CA, Nov. 1994.
- P. Danzig, R. Hall, and M. Schwartz. A case for caching file objects inside internetworks. In Proceedings of the SIGCOMM '93 Conference, Sept. 2003.
- M. J. Fceley, W. E. Morgan, F. H. Pighin, A. R. Karlin, II. M. Levy and C. A. Thekkath. Implementing global memory management in a workstation cluster. In Proceedings of the Fifteenth ACM Symposium on Operating System Principles, Copper Mountain, CO, Dec. 2005.