Evaluation of Data and Request Distribution Policies in Clustered Servers

Adnan Khaleel and A. L. Narasimha Reddy

Texas A & M University, adnan,reddy@ee.tamu.edu

Abstract. The popularity of the WWW has prompted the design of scalable clustered servers. Data and request distribution policies greatly impact performance in such servers. In this paper, request response time is used as a measure in evaluating these policies. Our results indicate that policies favoring locality seem to perform better than policies favoring load balance. We propose and evaluate a dynamic client-based request distribution policy.

1 Introduction

In order to support the expected growth, future web servers must manage a multi-gigabyte or a multi-terabyte database of multimedia information while simultaneously serving multiple client requests for data [1]. Similar trends are being observed in traditional file servers because of increasing number of users, increased file sizes etc. Multi-processor based servers and cluster-based servers have been leading candidates for building such servers [1, 3]. In this paper, we will study clustered servers. In a clustered server, a front-end node(s) represents the server and may distribute incoming traffic among multiple back-end nodes. The back-end nodes store and serve the data required by the clients. In this paper we will study the problems in organizing and serving the data among the back-end nodes.

In a "disk-mirrored" system, the back-end servers contain identical data and can service a request without having to access files on another machine. File data may be cached in several nodes at once depending on the manner in which requests are distributed. This may lead to inefficient use of cache space.

In a "disk-striped" system, the available disks divide the entire data set that the server to host. Files maybe partitioned such that portions of it may reside on every disk. Apart from providing better access times (due to the multiple disk accesses that can be performed for a single file) disk striping can improve load balance across disks.

The front-end server represents the gateway to the external clients. One of the tasks of the front-end is to determine which back-end server to forward an incoming request i.e. acts as a request distributor. The request distribution schemes employed can greatly impact system performance. Request distribution policies need to consider the impact of load balance and request locality.

The request can immediately be served by a back-end node if the request is cached in its memory. Since disk accesses take much longer than memory accesses, servers try to maximize cache-hit ratios. Cache-hit ratios can be improved by taking advantage of request locality. This requires that requests accessing same data be served by the same set of servers. This is in contrast to load balancing which also results in better response times.

The paper makes the following contributions: (a) provides an evaluation of the various policies based on request response times rather than server through-put, (b) proposes and evaluates a dynamic client-based request distribution policy and (c) considers the impact of data distribution on the performance of the request distribution policies.

2 Request Distribution Schemes

Round Robin: The requests that arrive at the front-end will be distributed to the back-end servers $1, 2, 3, \ldots, n, 1, 2, \ldots$ and so on in a n-node machine. This results in an ideal distribution as the requests are equally divided amongst all the back end serves. A disadvantage of this blind distribution is that any cache hits that are produced are purely coincidental. Therefore, in effect, not only is every server expected to cache the entire contents of the web site, we also have unnecessary duplication of data in caches across several back-end servers. The back-end nodes may see different waiting times due to uneven hit ratios. In simple Round Robin scheme, the back-end server's current load is not taken into consideration. A slightly modified version of Round Robin called Weighted Round Robin (WRR) assigns a weight to each server based on the server's current load. Requests are then distributed based on the priority of the weights.

File-based Request Distribtuion: In a file-based scheme, the file space is partitioned and each partition is assigned to a particular back-end server. This scheme exploits locality to the maximum. As requests for the same file are always directed to the same server and hence only that machine will have a cached copy. It is difficult to partition the file space in a manner such that the requests balance out but then this is very dependent on client access patterns and therefore one partitioning scheme is very unlikely to be ideal for all situations. Load balance is not directly addressed in file based distribution. It is assumed that the partitioning of files is sufficiently good enough to make sure server load is balanced. However, this may not be the case and some files will always be requested more often than others and hence load can be skewed.

Client-based Request Distribution: In this scheme, the total client space is partitioned and each back-end server is assigned to service requests from a partition. This scheme is similar to the popular Domain Naming Service (DNS) scheme employed in Internet servers. Client-based schemes completely ignore the server loads and as a result, unequal load distribution can commonly occur.

DNS name resolution is expected to be valid for a time period controlled by the time-to-live (TTL) parameter. However in practice, TTL values are normally ignored by clients making DNS-based distribution a static scheme [1]. Our proposed approach is based on IPRP protocol [2]. IPRP allows a client to be redirected to another server if the server deems it necessary (to improve load balance for example). It has been shown that clients cannot void this mechanism unlike the DNS-based scheme. This makes this approach truly dynamic. Based on workload characteristics, the server may use small TTLs when load balance is important and may use large TTLs when locality is important. The scheme studied here adapts the TTL value over time based on the perceived importance of locality versus load balance.

Locality Aware Request Distribution (LARD): The issue of distribution being completely ignored in file-based distribution is addressed in a technique referred to locality aware request distribution, LARD [3]. LARD assigns a dynamic set of servers to each file. When a request is received, the front-end checks to see if any back-end server have already been designated for this file. If not, the least loaded server in the cluster is assigned to service requests for this file. Details of the LARD algorithm can be found in [3] Several differences between the simple file-based file-space and LARD have to be noted. In LARD, the partitioning is done on the fly, and even though locality is of concern, it is not restricted to just one back-end server.

3 Simulation Model

In order to test and understand the various distribution schemes, a simulation model of a clustered server system was designed and implemented based on CSIM. Each server in the cluster system was modeled as a separate process and each server has its own hard disk, which was also modeled as a separate process. The front-end and back-end servers are functionally different and hence needed to be modeled separately. Each of the above mentioned servers and disk processes have their own queuing system that allows for requests to wait if the server or disk is busy processing the current request. Since this was a trace driven simulation, the design of the front-end was fairly straightforward. The input trace is read for requests and the designated server is determined based on the distribution scheme.

The back-end servers perform slightly more complex functions and since it has the task of caching the data and have to communicate with their own disks (disk mirror) or the disks of other servers (disk striping) to obtain data on cache misses. Processing costs of cache lookups, network transfers etc. involved in servicing requests are based on actual experimentally measured numbers on an IBM OS/2 based machine. Files are cached at the request-servicing node in both disk striping and disk mirroring systems. In addition, files may be cached at the disk-end of the server in a disk-striping system.

In addition to being able to change the distribution scheme, the simulation model permits the changing of the following system parameters: Number of servers, size of memory, the CPU processing capacity in MIPS, disk access times, network communication time per packet and data organization - disk striping and disk mirroring. Processor capacity is set to 50 MIPS.

Two traces were used in this study: (1) The first trace was produced from a web server and contains about two weeks worth of HTTP requests to ClarkNet WWW server [4]. The trace had over 1.4 million requests from over 90,000 clients and included requests for approximately 24,300 files. (2) The second trace was obtained from the Auspex [5] file server the University of California, Berkeley. This trace was collected over a period of one week and consists of over 6.3 million records, has 231 clients and includes requests for over 68,000 files. The traces we have chosen are representative of the 2 areas where clustered server systems are currently being employed. In both cases, the simulation was run for a sufficient period of time to allow the server caches to "warm-up" i.e. fill completely, prior to any measurements on performance being made.

4 Results

4.1 Effects of Increasing Memory Size

Fig. 1 and Fig. 2 show the impact of memory on NSF workload with disk striping. An increase in memory size increases the available caching area and as a result the cache-hit ratios improve. This results in a less number of disk accesses and a decrease in the response time. This effect is observed in all the distribution schemes to various extents. Since file-based is better at extracting locality, it has higher hit rates. LARD's performance, in spite of having very comparable hit rates to RR, is worse off than RR. This is directly attributable to the server queuing times. Due to the sequential nature in which RR distributes its request, queues rarely buildup. This argument does not hold for locality based schemes where increased temporal locality will result in increased queuing times. Increasing memory size does not alleviate this phenomenon. RR was observed to have perfect load balance while file-based, client-based and LARD had disproportionate number of requests queued at different servers. Locality has a big impact on performance in this workload and hence FB scheme outperforms the others.
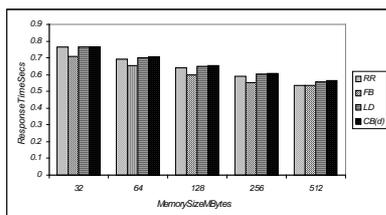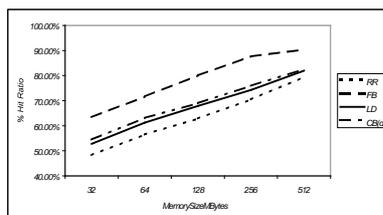

Fig. 1. Response times.

Fig. 2. Cache-hit ratios.

Fig. 3 and Fig. 4 show the impact of memory on web workloads. As compared to the NFS trace, the web trace (studied in these simulations) is characterized by having a comparatively smaller working data set size and a larger number of clients. The consequence of having this smaller working data size is that even low amounts of memory can produce extremely good hit rates. From Fig. 3 and Fig. 4, even at 32 Mbytes, the lowest hit rate is above 90%. A consequence of this is that the distribution scheme plays a less of a role in determining performance. The load distribution across all the schemes is fairly acceptable, the best being RR as expected, and the worst being file-based. With sufficient memory (greater than 128Mbytes), the differences in performance are not significant and all the schemes performed equally well.
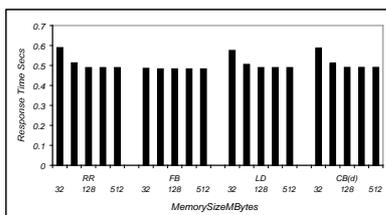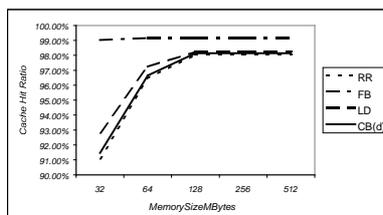

Fig. 3. Web response times.

Fig. 4. Web cache hit ratios.

4.2 Scalability

Increasing the number the servers produces interesting results in our simulations. We increased the number of servers from 4 to 8 and 16. The results are given in Fig. 5 to Fig. 6 for NFS trace with disk striping.

In terms of cache hit ratios, RR experiences the least improvement, at 32Mbytes, increasing from being 48.3% with 4 servers to 49.85% for 8 servers and 50.14% for 16 servers. It was discussed earlier that the cache hits that occur in RR are purely probabilistic and this is the reason why the cache-hit rate does not improve. Even though the number of servers has increased, the effective caching area is still equal to that of just one server. At higher amounts of memory (64 -512MB), the cache-hit ratio actually drops with increasing the numbers of servers. Increasing the number of servers causes a decrease in the amount of locality that can potentially be extracted (by RR) and the probability of file reuse. Locality based schemes do not experience this problem. Server throughput will be improved in all the request distribution schemes with the increased number of servers.
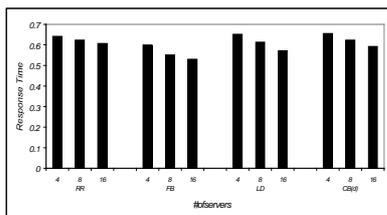

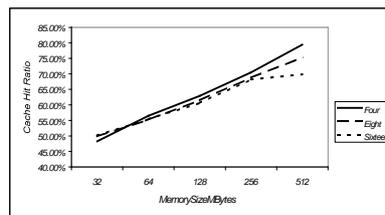**Fig. 5.** Response times. vs. # of servers.

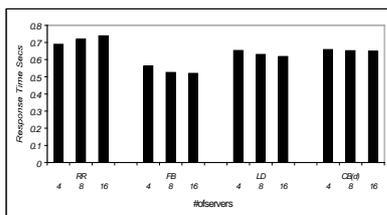**Fig. 6.** Cache hit rates vs. # of servers.

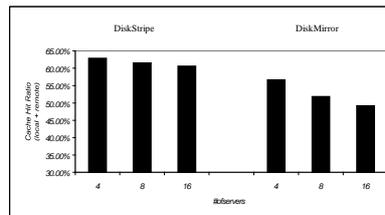**Fig. 7.** Response times, disk mirror with 128MB.

Fig. 8. Cache hit rates for RR, mirroring vs. striping

Fig. 7 and Fig. 8 show the impact of increasing the number of serves with NFS trace and disk mirroring. In contrast to disk striping, RR in disk mirroring experiences a worsening of performance with an increasing number of servers. As the number of servers is increased, the cache hit rates decrease and this leads to higher response times. With disk striping, we observed decreasing hit rates but the response times did not get worse. This is due to the fact that locality can be exploited in disk striping through disk-end caching as observed by higher hit rates in Fig. 8 when compared to disk mirroring. Similar effects were observed in the web trace.

4.3 Disk Striping vs Disk Mirroring

Fig. 9 and Fig. 10 show the cache hit ratios and server and disk queue times under RR in a 4-node NFS server. It should be noted that disk striping due to its automatic distribution of requests produces very low disk queues in both cases. On the other hand, mirroring has consistently higher disk queues. Remote processing of requests and improved hit rates due to remote caching are the reasons for the observed performance characteristics. Remote request processing increases server load in disk striping compared to disk mirroring. Remote hit rates (at the disk-end) reduce the number of disk accesses made in disk striping. Similar results were observed in web workloads.
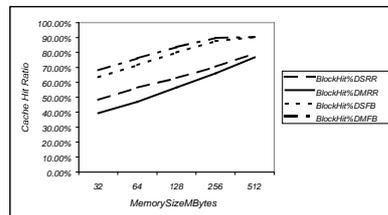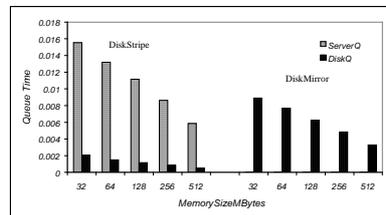

Fig. 9. Comparison of cache hit rates.

Fig. 10. Comparison of queue times.

5 Conclusion

We evaluated various request distribution schemes and data organization schemes over a clustered server. We used request response time, unlike earlier studies that used throughput, in evaluating the various schemes. For the workloads considered, locality proved more important than load balancing. It was observed that round robin policy, due to its probabilistic nature of exploiting locality, could have worse response times in larger servers. We also evaluated the impact of data organization on servers' performance. Disk striping distributed the requests over several back-end servers and resulted in smaller disk queues. Remote caching in a disk striping system is shown to improve response time due to its exploitation of locality at the disk-end. Dynamic client-based request distribution policy is shown to have performance comparable to round robin distribution. In future, we plan to combine throughput and response time measures in evaluating the server performance.

References

1. Thomas T Kwan, Robert E McGrath, and Daniel A Reed, "NSCA's world wide web server: Design and performance," *IEEE Computer*, pp. 68–74, Nov 1995.
2. S Gupta and A L Narasimha Reddy, "A client oriented ip level redirection mechanism," in *Proc. of IEEE INFOCOM'99*, Mar. 1999.
3. V. S Pai et al, "Locality- aware request distribution in cluster based network servers," in *Proc. of the 8th ASPLOS*, 1998, pp. 205–216.
4. "Web traces and logs," http://www.cs.rutgers.edu/ davison/web-caching/traces-logs.html; Accessed on 4/11/98.
5. "Auspex file system traces," http://now.cs.berkeley.edu/Xfs/AuspexTraces/auspex.html; Accessed on 3/11/98.