Coordinated Resource Management in a Replicated Object Server

Sanjay Ghemawat
Robert Gruber
James O'Toole
Liuba Shrira

Abstract

We propose several new techniques for resource management in a replicated object server. By coordinating cache and disk usage among the replicas, these techniques increase throughput and reduce fetch latency. *Cache splitting* speeds up fetches by avoiding redundant cache entries, effectively increasing the cache size. *Coordinated writing* coordinates disk writes to ensure that one replica is always available to service fetches. We investigate the performance of a replicated server using these techniques, and we present simulation results showing that these techniques provide substantial performance improvements across a variety of workloads.

1 Introduction

Traditional replicated server designs use multiple replicas to improve system availability and reliability. Although close communication between the replicas is required for transaction serialization, each replica independently manages its cache memory and disk resources. In this paper we propose new techniques that exploit a previously overlooked opportunity to enhance performance by having replicas coordinate use of caches and disks.

Authors' addresses: sanjay@lcs.mit.edu, gruber@lcs.mit.edu, otoole@lcs.mit.edu, liuba@lcs.mit.edu. Laboratory of Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139.

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-91-J-4136, in part by the National Science Foundation under Grant CCR-8822158, and in part by the Department of the Army under Contract DABT63-92-C-0012.

In a traditional replicated server, objects can reside in an arbitrary number of replica caches. *Cache splitting* keeps a segment in the main memory of one replica only and accesses it over the network when the segment is needed at another replica. Splitting speeds up fetches because it increases the effective cache size. Cache splitting can also speed up updates, because writing a modified object back to disk often requires reading from disk the segment on which the object resides.

Disk arms can be scheduled more efficiently if write requests are accumulated and applied together, a technique known as *batching*. Studies show that batching can provide significantly higher bandwidth than random access [12]. Nevertheless, batching can substantially delay client fetches, which might block behind batched writes [4]. Such varying delays can substantially reduce the effectiveness of client prefetching. *Write coordination* avoids this problem by ensuring that one replica is always available to service fetches, allowing the server to enjoy the benefits of batching without imposing a penalty on fetches.

Although we believe that these techniques are applicable to a variety of replicated server architectures, we explore them here in the context of a primary-copy server in the Thor persistent object system [8]. Thor clients fetch objects from the server into a local cache, compute locally, and then send any modifications to be committed at the server. As such systems scale, they tend to become I/O bound, so techniques for enhancing I/O latency and throughput are of substantial interest. By shifting load from the disk to the network and main memory, these techniques will become increasingly more effective as network and main memory speeds continue to improve faster than disk access times.

We simulated the operations of a multiple-client two-replica object server using a simplified hardware model. We evaluated the performance of two simple baseline configurations and compared them to three additional configurations that use combinations of our techniques. Our simulation results show that our techniques provide substantial performance improvements across a variety of workloads.

In the following sections, we introduce the basic replicated object server design (Section 2) and describe the coordinated resource management techniques (Section 3). We then introduce our simulation model (Section 4), explain the experimental workloads and test configurations (Section 5), and present simulation results that illustrate the value of the new techniques (Section 6). Finally, we discuss related research (Section 7) and our conclusions (Section 8).

2 A Replicated Object Server

This section describes our "baseline" replicated object server; an understanding of this baseline is necessary before we can present our new techniques based on the coordination of replica resources.

The baseline is derived from our work on Thor [8], a client-server object-oriented database system with replicated object servers. Three important features of Thor that we use in our baseline are optimistic concurrency control, object-based architecture, and primary copy replication.

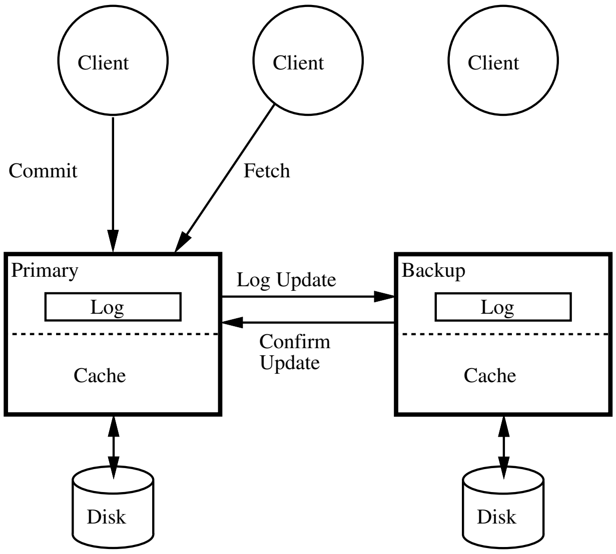


Figure 1: Clients talking to a primary copy server.

Fetching

Clients fetch objects to a local client cache as objects are read, update objects in their cache as they are written, and send back updated objects in a commit request to the server. The clients and the server together execute a concurrency control and cache consistency protocol that ensures all transactions are serializable and all client caches are "almost" up-to-date. Since we use an optimistic scheme, a client transaction may read an out-of-date value, in which case it will be aborted and restarted by the server when it sends its commit request. We assume the need to abort is detected at the server using a validation phase that does not require any disk accesses; see [1] for the details.

Figure 1 shows the structure of the system during normal operation. The server is implemented using a primary-backup-witness scheme. The witness is not depicted; it is not used in the normal case, and our techniques do not apply to the "failover" case where the witness has taken over the task of the backup.

Committing

Each replica has an in-memory log of committed updates. An important point is that committing a transaction does not require a disk write. All commit requests are sent to the primary, which logs a commit record containing the transaction's updates and forwards it to the backup's log. A transaction is considered committed once its commit entry is present in both logs — each machine running a replica has an uninterruptible power supply to prevent the simultaneous loss of both logs. This approach to fast primary copy commit was invented for the Harp replicated file system [9].

Installing

The system uses an object-based rather than a page-based architecture. Objects managed by the server are grouped into *segments*, and each replica has a *segment cache*; segments are the unit of disk transfer and caching at the server replicas. Clients, on the other hand, fetch and cache objects, and send object updates to the primary in a commit request; these object updates are placed in the log, and are applied to the appropriate segments when the transaction commits; we call this update process *installation*. Note that an install may require a disk read if the segment to be updated is not currently cached. This is in contrast with page-based systems, where entire dirty pages would be logged, in which case an install would never require a disk read. This is an important point, and throughout the paper we are careful to distinguish between disk reads initiated by a client fetch request and reads initiated by an install.

We assume the install for a transaction occurs at a replica as soon as it knows the transaction has committed. Installs are synchronized with fetch requests, to ensure that fetches retrieve the current committed object state. Note that the primary and backup are already synchronized with respect to transaction commits (via the logging protocol) thus either replica can be used to handle fetch requests. We discuss this further below.

Writing

Installs modify segments in the cache, but they do not immediately cause the updated segments to be written to disk. If the updates from more than one transaction are installed in a segment before it is written, we have saved some writing compared to a system that writes each segment as soon as it is modified by an install; we say there has been *write* *absorption*. Write absorption is a very important means of reducing disk traffic; we would like to wait for as long as possible before writing a dirty segment. However, such waiting causes a problem: the memory space allocated to the log can fill, requiring *log truncation*.

The replicas run a background process that detects when the log is getting full, initiates a set of segment writes, and truncates the log. The dirty segments containing the oldest committed updates are written to disk; as these writes complete, the tail end of the log (the entries for the updates that are now on disk) can be discarded. This process also of course cleans the written segments and ensures the cache does not become entirely dirty. We assume this process uses *write batching*: it writes out a large enough batch of dirty segments to permit efficient disk arm scheduling.

Balancing

We have now described our baseline system, which we use to discuss our new techniques. When comparing the different schemes, we also consider a modified baseline system that uses a form of load balancing which we call *fetch balancing*: rather than sending all fetch requests to the primary, clients can issue fetch requests to both replicas. Each replica therefore processes some of the fetches and performs some of the reads required. Fetch balancing is similar to the scheme described in [6] that sends read-only queries to the backup in a replicated relational database.

3 New Techniques

In this section we describe our techniques for coordinating the use of cache memory and disk arms at the server replicas. We also discuss the corresponding performance tradeoffs.

3.1 Coordinated Writing

The disadvantage of write batching is that very large read delays will occur when a disk read is scheduled while a large disk write is in progress. In this situation, better read performance could be obtained by forwarding the read request to the other replica. If the other replica is not also performing a large write, the result will be a much shorter read delay, in spite of the additional network roundtrip.

Ideally, read operations would never be delayed by large disk writes; we can ensure this by ensuring that only one replica is writing at any time. As illustrated in Figure 2, the primary and backup replicas can perform their write batching in an alternated schedule. A simple coordination protocol can be carried out by the two replicas; we call this *coordinated writing*.

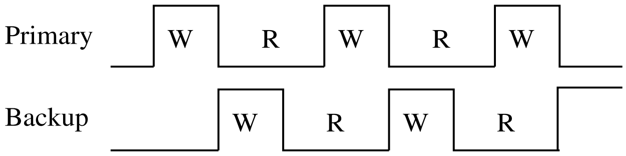


Figure 2: Coordinated writing

3.2 Cache Splitting

With autonomous cache management, there may be a high degree of overlap between caches. The disadvantage of having the same segments cached at both servers is that fewer total segments of the database are in cache memory. If we can coordinate the contents of the caches to avoid overlap, we may be able to achieve a better overall hit rate. This suggests the idea of *cache splitting*.

With cache splitting, each database segment is assigned a designated “home” cache. In its home cache, a segment is treated fairly by the cache replacement algorithm. In its “foreign” cache, a segment is severely discriminated against by the cache replacement policy. This technique will tend to reduce the degree of cache overlap.

To benefit from cache splitting, clients should direct their fetch requests to the home replica for the fetched segment. When a replica installs an update, the segment should be obtained from its home cache if it is not already cached locally. These policies will further encourage cache segregation and will usually ensure that a currently cached segment will not be read from disk. Figure 3 illustrates how client fetch operations are routed to the appropriate replicas. As shown here, the primary replica preferentially caches type A segments and the backup cache prefers type B segments.

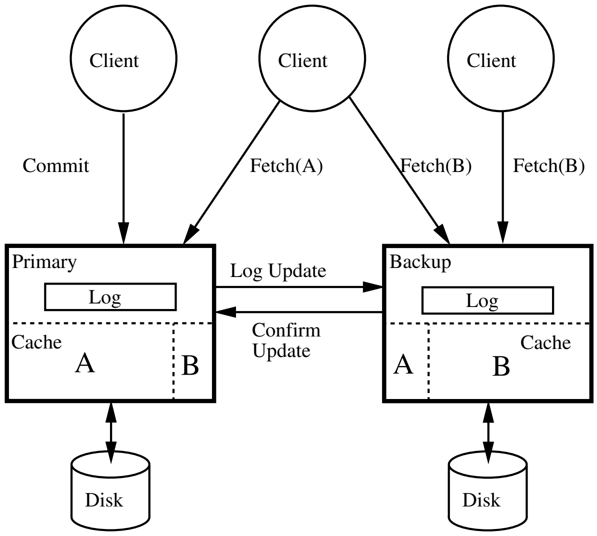


Figure 3: Cache splitting

When a writing transaction commits, both replicas must perform an update installation for the modified segment. Without cache splitting, both replicas might read the segment from disk in order to perform the installation. With cache splitting, the segment will be read from disk at most once, because both replicas will obtain the segment from the segment's home cache. (We also note that if the installation is performed in the home cache before the segment is requested by the foreign replica, then the installation need not be repeated. Discussion of this additional optimization and several related techniques is postponed to future work.)

3.3 Performance Tradeoffs

This section discusses the performance tradeoffs involved in the coordination techniques described above.

Network vs. Disk or Memory

Both of the coordination techniques we have introduced use read forwarding to shift read operations between replicas. Forwarding reads increases the inter-replica network load. In addition, coordinated writing increases inter-replica network load because extra messages are exchanged for coordination purposes.

We believe that trading network bandwidth for improvements in cache memory and disk utilization will probably always be advantageous because network latency will generally be superior to disk latency. We do not expect network saturation to be a limiting factor: if necessary, a dedicated network can be used for inter-replica communication to avoid contention with client-replica message traffic. Such a network is also useful to improve availability.

There is one case where cache splitting could degrade system performance. If all of the hot segments of the database fit easily into a single replica cache, then cache splitting will segregate these hot segments between the two caches. As a result, installations on these segments will require one replica to read the segment from the other replica. This will delay many installations by a network round-trip.

However, we do not expect this installation delay to affect system performance. Installations are performed asynchronously in our replicated object server design. Therefore, it seems that the worst possible result of a delayed installation is to slightly delay the initiation of a batched disk write. This could only be important if maximum write utilization of the disk drive were of such importance that essentially all available disk bandwidth was being dedicated to writing.

In general, the coordinated resource management techniques we have introduced are directed towards improving overall performance in systems where read latency is an important concern. We believe that this is a sensible expectation for future replicated object servers. We are much less interested in workloads that use disks primarily (or only) for writing, and believe that such workloads are also not likely.

Write Capacity vs. Read Latency

The coordinated writing technique is intended to provide better fetch response time, but restricts the maximum available disk write bandwidth. In particular, the worst-case fetch response time will be much lower because no disk reads will wait for the completion of a large write batch. However, because only one disk is permitted to write at a time, the maximum available write bandwidth is half as great as in an uncoordinated system.

It seems possible that under a sufficiently write-intensive workload a standard primary copy method could perform better than a system using coordinated writing. If the total disk write traffic in the primary copy method exceeds 50% of the available disk bandwidth, then coordinating writing could slow down the system. For a workload to utilize the disk so heavily for writing, the disk write load must exceed the disk read load.

If each write has a corresponding installation read, this is not possible. Therefore, such a workload must involve a large number of writes to "hot" segments that are always in the cache. In serving such a workload, it seems likely that very high write-absorption could be achieved, which would further reduce the actual disk write load. In addition, because write batching substantially reduces the average cost of writing, a workload that suffers under coordinated writing would require an extremely high write ratio. For these reasons, we do not expect to observe such workloads in practical large-scall systems.

4 Simulation Model

In order to study the performance of our coordination techniques, we constructed and simulated a simple model of a replicated object server system. The model highlights the critical parameters and ignores the secondary ones. Our intention is not to predict the actual performance of the system but to provide insight into the relative merits of our coordination techniques.

The model consists of a number of identical clients connected by a network to a replicated database server. The database server maintains a fixed size database that consists of a fixed number of disk segments.

The components of the simulated model are described below. The system parameters that control the behavior of these components are listed in Table 1. The hardware parameters reflect to some degree our assumptions that future processors and networks will improve significantly relative to disk drives.

Clients

The simulated clients each contain a cpu and a local object cache. The simulator does not model the contents of the client cache, but is parameterized by the client cache hit rate. The clients execute a sequence of transactions. During each transaction, the client accesses a single object in the database. If the object is not in the client cache, the client fetches the object from an object server. After the client obtains the object, it computes for some time and possibly modifies the object. Then the client commits the transaction by sending a message to the primary server.

After the client receives confirmation of the commit, it immediately begins the next transaction. This simplistic model of the clients is sufficient for our purposes because we are investigating the performance of schemes that run at the server and are therefore mostly interested in the workload presented to the server. Our simulation results may not be realistic in absolute terms: a real system would probably support many more clients because clients would do more fetching and computation in each transaction.

Network

The simulator models a network without contention. The network has unrestricted bandwidth and offers a fixed message delivery latency. We chose to ignore network contention because the strategies under consideration do not affect network traffic except between the two server replicas. If network traffic between the server replicas were a limiting factor in the performance of a replicated system, we would expect it to be economical and feasible to provide a dedicated high-bandwidth network channel between the replicas. Thus, we do not expect the network model to significantly affect the results of our experiments.

Server

The primary and backup replicas service fetch and commit requests from clients by reading and writing relevant database segments that live on attached disks. Each replica maintains an LRU cache of database segments. This cache is used to speed up the handling of client requests.

We model concurrency control costs at the server with a simple parameter that expresses the time required to validate each transaction. Aborts are not explicitly modeled in our system. If transactions were aborted and restarted this would simply result in extra read traffic to the server. Therefore a workload that generates a lot of aborts can be simulated with a workload that generates extra reads.

We do not explicitly model the transaction log stored at each replica. In evaluating our techniques, we expect the only significant influence of the log on system operations to be that the log size must be bounded. We model a bounded log size by limiting the number of dirty segments in the replica cache. Limiting the number of dirty segments also restricts write absorption in our experiments and ensures that the results are not biased by variations in write absorption.

The implementations of the various cache and disk management strategies used in our simulated server are described in Section 5.2.

Database	
Database size	100,000 segments
Segment size	4 Kbytes
Clients	
Cache hit ratio	80%
Processing time per transaction	1 msec
Network	
Message latency	1 msec
Processor time per network send	100 μsecs
Server	
Transaction validation time	5 μsecs
Modification installation time	1 msec
Write when this many dirty segments	20% of cache size
Write this many segments at a time	1000
Disks	
Time to read/write one segment	30 msecs
Time to write 1000 segments	4 seconds

Table 1: System Parameters

Disk

The simulator models the disk as a simple queue that handles disk operations in the order of their insertion into the disk request queue. An individual disk operation is completed in a fixed amount of time. If a large number of disk operations are submitted together to the simulated disk, then these operations are completed much faster than they would if they were submitted individually.

This simplistic disk model is intended to capture the essential aspect of disk drive performance that is relevant here: when a large number of operations are outstanding, the disk driver or scheduler can order the operations to obtain approximately one-half the raw disk transfer bandwidth available [12].

5 Experiment Setup

We designed and ran a series of experiments to evaluate the coordination techniques presented in Section 3. We compared the performance of several experimental configurations on a variety of workloads. In this section, we describe the configurations and the workloads used in our simulations.

5.1 Workloads

The parameters that define the workloads used in our experiments are listed in Table 2. We have two types of access patterns in our workloads — *uniform* and *skewed*.

Workload Parameters	
Number of clients	$2 \ldots 32$
Server cache size	5,000 or 25,000 segments
Access pattern	Uniform or skewed
Skew parameters	
Hot region size	10,000 segments
Hot access probability	90%
Write probability	10%

Table 2: Workload Parameters

Under a uniform workload, a client picks the segment to reference with uniform probability from the entire database. Under a skewed workload, the database is partitioned into two regions, hot and cold. A referenced segment is selected uniformly from the hot region with probability *hot-access-probability* and from the cold region with probability (1 − *hot-access-probability*). Each client transaction modifies the segment it references with probability *write-probability*.

5.2 Experimental Configurations

This section contains a brief description of the experimental configurations used in our simulations. All of these configurations use *write grouping* to increase write absorption and disk throughput.

Standard Uncoordinated

The clients send all fetch requests to the primary replica. The backup replica is used only by the transaction protocol to log and install updates to the database. The standard uncoordinated configuration does not use any of the optimizations discussed in Section 3.

Balanced Uncoordinated

In the balanced uncoordinated configuration, the standard baseline configuration is augmented with random fetch balancing. Clients choose randomly between the primary and backup replicas when sending fetch requests. We added this configuration to our simulation suite because part of the advantage of cache splitting is due merely to improving load balance at the replicas.

Split Uncoordinated

Cache splitting is used to decrease the overlap between the caches at the two replicas. The clients send a fetch request for a segment s to the *home* cache for segment s.

We implement cache splitting in our simulations simply by making one replica contain the home cache for odd-numbered segments and the other replica contain the home cache for even-numbered segments.

Balanced Coordinated

The clients behave as in the randomly balanced configuration. The replicas use *coordinated writing* and *read forwarding* to prevent reads from getting stuck behind large disk writes.

Split Coordinated

This configuration combines coordinated writing and cache splitting.

Other configurations

We simulated many other combinations of the server configurations, simulation parameters, workloads, and write policies. We found the other combinations useful in verifying our understanding of the configurations presented above. However, we see no reason to clutter our presentation with the additional configurations. Readers interested in obtaining the complete simulation results or the simulator itself should contact the authors.

6 Experiments

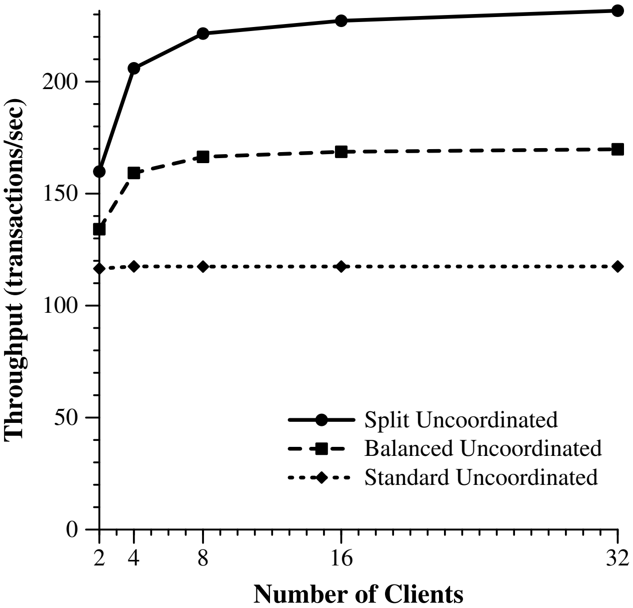
In order to understand how the performance of the experimental configurations compare to each other, we simulated each configuration on a variety of workloads. We randomly generated several transaction traces for each workload. Each configuration was run until there were no significant variations in the results.

We measured transaction throughput, average fetch latency, maximum fetch latency, and a variety of resource utilization metrics. These measurements were always sampled over intervals corresponding to five write-batch cycles (i.e. both the primary and the backup replica each wrote five batches of 1000 segments to its disk).

In the sections that follow we present some simulation results that illustrate the relative performance achieved by the various experimental configurations. We present these results in terms of "read-bound" and "write-bound" workloads. However, the reader is cautioned to take care in interpreting these terms: because all configurations perform batched writes, the system typically moves among distinct modes in which a particular replica uses its disk exclusively for reading or writing. Therefore, even a "write-bound" workload spends some portion of the time operating in a read-only mode.

6.1 Read-Bound Workloads

We expect the effects of cache separation to be easier to observe in read-bound workloads, so we first examine the uniform workload with a small cache. Figure 4 shows the throughput of the non-write-coordinated configurations for various numbers of clients. The table accompanying the figure shows the cache hit ratio and average fetch latency for the eight client workload. We chose eight clients because most throughput curves start flattening out with more clients, and the fetch latency rapidly seems unacceptably large.



Metrics for eight clients

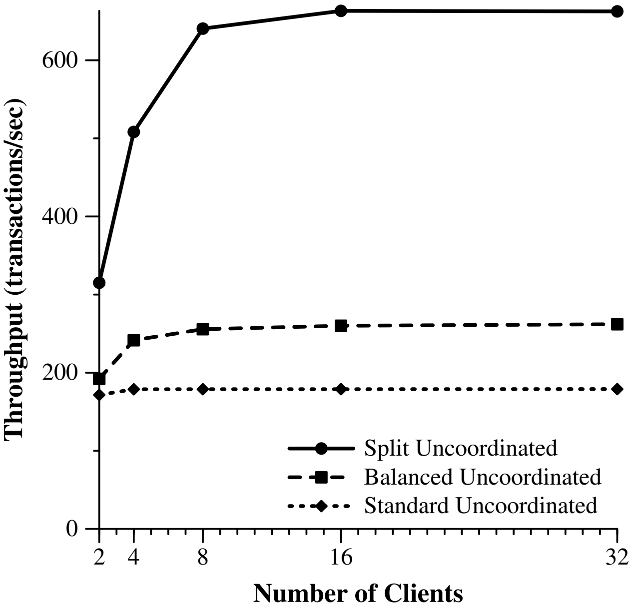
Scheme	Cache Hit	Avg. Latency
Split Uncoordinated	9%	164 ms
Balanced Uncoordinated	5%	224 ms
Standard Uncoordinated	5%	323 ms

Figure 4: Uniform workload in a small cache.

The split uncoordinated configuration has much better performance than the other configurations. We expect the cache hit ratio in split uncoordinated to be approximately double that of the other configurations shown here. However, because the cache hit ratio is so low, the cache miss ratio is not dropping very much in relative terms (from 95% to about 90%). The improvement shown in Figure 4 is too large to attribute to such a small decrease in fetch disk reads.

In fact, the improvement is due to a twofold reduction in installation reads. In the non-splitting configurations, both the primary and the backup replicas perform the installation reads independently. Splitting the installation reads significantly reduces the number of disk reads required to install updates. Note that this effect is specific to replicated object servers because page-based systems such as file servers typically do not require installation reads.

To maximize the effect of cache splitting in a read-bound configuration, we examine the skewed workload in a small cache. The small cache holds 5,000 segments and the skewed workload makes 10,000 segments hot, so cache splitting should roughly double the cache hit ratio from about 45% to about 90%.



Metrics for eight clients

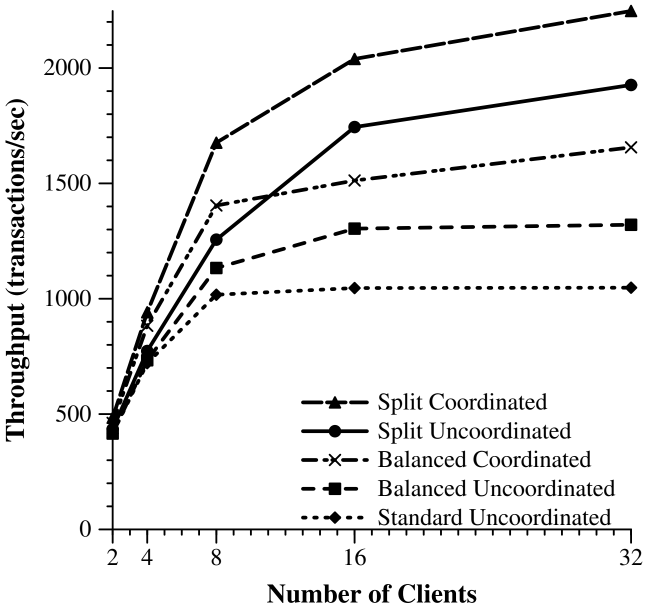
Scheme	Cache Hit	Avg. Latency
Split Uncoordinated	75%	46 ms
Balanced Uncoordinated	40%	140 ms
Standard Uncoordinated	40%	206 ms

Figure 5: Skewed workload in a small cache.

As shown in Figure 5, the split uncoordinated configuration now has much higher throughput and much smaller average fetch latency. As in Figure 4, the balanced configuration performs better than the standard configuration. However, now that the actual cache miss ratio is dropping from about 60% to roughly 25%, we observe a large increase in throughput. As before, part of the improvement is attributable to avoiding redundant installation reads. However, now the increased cache hit ratio is a significant factor because many fewer client fetch operations and update installations require disk access.

6.2 Write-Bound Workloads

Coordinated writing should dramatically reduce the maximum fetch latency observed by clients because fetch misses will never wait for the completion of a large disk write. We simulated all experimental configurations using a more write-intensive workload. For illustration purposes, we chose the skewed workload in a large cache. Because the workload is skewed and each cache now holds 25,000 segments, cache hit ratios are now above 90% in all configurations, and disk writes become much more significant.



Metrics for eight clients

Scheme	Cache Hit	Latency	
		Avg.	Max.
Split Coord.	94%	7 ms	203 ms
Split Uncoord.	94%	14 ms	4013 ms
Balanced Coord.	91%	11 ms	285 ms
Balanced Uncoord.	91%	18 ms	4054 ms
Standard Uncoord.	91%	22 ms	4159 ms

Figure 6: Skewed workload in a large cache.

Figure 6 shows the throughput of all simulated configurations, the cache hit ratio, the average fetch latency, and the maximum fetch latency. The results show that the coordinated writing configuration provides much better maximum fetch latency. We can also see an increase in throughput due to the fact that clients are able to continue fetching and committing transactions while the replicas write.

Figure 6 shows that cache segregation alone also offers an improvement. This improvement is due partially to the reduction of installation reads (as in the small cache results), but also due to a decrease in the cache miss ratio from approximately 9% to about 6%. We can also see that as more clients are added to the system, the value of cache splitting appears to exceed the value of coordinated writing. Of course, the best performing configuration is split coordinated, because it benefits from the combination of both techniques.

We also simulated these configurations using other access patterns and a variety of write ratios. The results are generally similar to those above. As expected, in uniform access patterns where disk reads are more important, the advantage of cache splitting is more significant.

7 Related work

There is a substantial body of work in the literature concerning the operation of replicated servers. Recently, replicated server design and performance have attracted much attention [9, 7, 3, 10, 2].

Previous efforts to evaluate the utilization of the combined resources of both replicas has focused primarily on improving the load balance between the primary and the backup replica. One approach to load balancing in a replicated database design is to execute read-only queries at the backup [6]. This technique improves server throughput by increasing processor and disk arm utilization at the backup.

A different approach to load balancing was implemented in the Harp file system [9]. In Harp, data is statically partitioned between file systems and each active site serves as the primary replica for one file system and as the backup replica for another. In a transactional system, however, such data partitioning introduces extra distributed commits when a single transaction accesses data in more then one partition.

In contrast to our work, none of the above designs attempt to provide global management of the contents of caches or to coordinate the conduct of disk I/O operations. We are not aware of other work in the replicated servers area that considers the aggressive coordinated resource management that we propose. Franklin, Carey, and Livny [5] describe a global cache management technique in a database system. The pioneering work of Polyzois, Bhide, and Dias [11] studies the performance of write coordination in a mirrored disk system.

8 Conclusion

In this paper, after presenting a baseline replicated object server design, we proposed several new coordination techniques that enable the server to utilize cache memory and disk bandwidth more efficiently.

The cache splitting technique helps segregate cache contents to avoid redundant storage of data. Client and server operations are selectively routed to the cache that will more likely contain the data. Cache splitting increases the effective cache size of a two-replica design and also avoids redundant reads for update installations.

The coordinated writing technique reduces maximum fetch latency. By always keeping one disk arm available to service read requests, the server can offer much better fetch latency even when a large disk write is in progress.

We built a simplified simulation model that captures important expected characteristics of future scalable replicated object server systems. We presented simulation results that illustrate the value of cache splitting and coordinated writing both alone and in combination. The results show that simple coordinated resource management techniques can offer substantial performance improvements. In particular, in systems where good fetch response time is important, cache splitting and coordinated writing reduce both the expected and the maximum fetch response time, while increasing total system throughput.

Although we focused on an object-based server that uses optimistic concurrency control and primary copy replication, we believe our techniques are general and apply to page-based systems, to systems that use locking rather than optimism, and to systems that use other replication techniques.

References

[1] Atul Adya. A distributed commit protocol for optimistic concurrency control. Master's thesis, Massachusetts Institute of Technology, February 1994.

[2] D. Agrawal and A. El Abbadi. Resilient logical structures for efficient management of replicated data. Technical Report TRCS 92-04, Department of Computer Science, University of California, Santa Barbara, CA 93106, 1992.

[3] A. Bhide, E. Elnozahy, and S. Morgan. A Highly Available Network File Server. In *Winter 1991 USENIX Conference*. USENIX Association, January 1991.

[4] S. Carson and S. Setia. Analysis of the periodic update write policy for disk cache. *IEEE Transactions on Sowtware Engineering*, 18(1):213–226, January 1992.

[5] M. Franklin, M. Carey, , and M. Livny. Global memory management in client-server dbms architectures. In *Proceedings of 18th VLDB Conf.*, 1992.

[6] H. Garcia-Molina and C.A. Polyzois. Processing of read-only queries at a remote backup. Technical Report CS-TR-354-91, Department of Computer Science, Princeton University, December 1991.

[7] H. Garcia-Molina and C.A. Polyzois. Evaluation of remote backup algorithms for transaction processing systems. In *ACM SIGMOD International Conference on Management of Data*, pages 246–255, San Diego, CA, June 1992.

[8] B. Liskov, M. Day, and L. Shrira. Distributed object management in Thor. In M. Tamer Özsu, Umesh Dayal, and Patrick Valduriez, editors, *Distributed Object Management*. Morgan Kaufmann, San Mateo, California, 1993.

[9] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams. Replication in the Harp file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1991.

[10] C. Mohan, K. Treiber, and R. Obermack. Algorithms for the management of remote backup databases for disaster recovery. IBM Reasearch Report RJ 7885R, IBM Almaden Research Center, San Jose, CA, June 1990.

[11] C. Polyzois, A. Bhide, and D. Dias. Disk mirroring with alternating deferred updates. In *Proceedings of 19th VLDB Conf.*, 1993.

[12] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *Proceedings of Winter USENIX*, 1990.