A Comparison of Cache Performance in Server-Based and Symmetric Database Architectures

Avraham Leff, Calton Pu, Frederick Korz

Department of Computer Science Columbia University, New York, NY 10027

CUCS-016-90

Abstract

We study the cache performance in a symmetric distributed main-memory database. The high performance networks in many large distributed systems enable a machine to reach the main memory of other nodes more quickly than the time to access local disks. We therefore introduce *remote memory* as an additional layer in the memory hierarchy between local memory and disks. In order to appreciate the tradeoffs of memory and cpu in the symmetric architecture, we compare system performance in alternative architectures. Simulations show that, by exploiting remote memory (in each node's cache), performance improves over a wide range of cache sizes as compared to a distributed client/server architecture. We also compare the symmetric model to a centralized-server model and parameterize the performance tradeoffs.

1 Introduction

In large distributed systems, vast amounts of hardware are connected together by fast networks ranging from local-area networks to the envisioned National Research and Education Network. These networks introduce a new level in the memory hierarchy -- main memory accessed through the network -- whose access time may be significantly faster than that of local disks. We call this *remote memory*. Disk access performance has been limited by seek time, stuck for decades in the range of a few tens of milliseconds. In contrast, current remote procedure call (RPC) implementations over Ethernet take only a few milliseconds round trip [3]. Moreover, the bottleneck in communications protocols is CPU and software overhead. With RISC technology doubling CPU speed every few years, we can expect even better remote memory access times in the near future. Furthermore, faster gateways and higher bandwidth will steadily bring down the communications overhead over wide-area networks.

In these large distributed systems, however, one can easily fail to realize the potential of high performance suggested by the redundancy inherent in such systems. For example, many distributed software systems are organized according to the client/server model. Its main advantage is simplicity, since the server resembles a centralized system. However, the asymmetry in the client/server model severely restricts resource sharing. For instance, useful data in a client buffer are inaccessible to other nodes in the system. From the client/server point of view, this problem disappears if we have enough buffer space in the server. This answer, however, begs the question of how to better utilize the ample memory resources in a large distributed system.

In this paper, we study a symmetric architecture in which all nodes of the database system are accessible to one another. Concretely, we distribute data and load, blurring the distinction between clients and servers. In particular, all nodes in the database can ''serve'' requests if their buffers contain the requested data item. We first contrast this architecture with the distributed client/server model where clients may only make requests for data to the "owner"/server of the object. We show in this paper that the large amount of remote memory available in the symmetric architecture makes a significant contribution to system performance gains. We next compare the symmetric architecture to a centralized-server model in which a a single, powerful, central server (with sole control of the database) handles all interactions with the database. The central-server model has apparent advantages: we therefore parameterize the memory and CPU tradeoffs between this and the symmetric distributed database model.

2 Simulation Study

We have created a simulation program to evaluate the performance of (1) a symmetric distributed main-memory database (SDMD), (2) a distributed client/server database (DCSD), and (3) a centralized-server database. Our simulation model abstracts details that do not affect the performance issues that we are investigating.

The simulation is implemented on top of the SMPL subsystem [5]. The model's building blocks are the CPUs, the network, main memory and disk resources: higher level simulation activities are composed of requests to these simpler resources. For example, a remote disk read is composed of a request to the remote node, a local disk read there, and a reply to the requester. Each resource has its own priority queue, which is serviced according to a FIFO disciple. Software costs such as cache access and maintenance are charged in terms of CPU service time and the number of network messages.

The simulation model has a fixed skeleton (which models the SDMD, distributed client/server, and centralized-server system structure), and three movable parts, which are the components of the redundancy management system (described in sections 2.3, 2.4, 2.5),-- i.e., the location algorithm, the replacement algorithm, and the consistent update algorithm respectively. We first describe the skeleton architecture, and then discuss the components of the cache management system.

2.1 The Architectures

The hardware model underlying the distributed databases is a collection of computer systems connected by a medium- to high-performance communications network. Each computing system has one or more modern microprocessor CPUs, relatively large main memory (e.g. 16 MBytes per CPU), and sufficient secondary storage (e.g. magnetic or optical disks) to hold a portion of a distributed data base. Such hardware bases are common in academia and industry.

In our simulation, we model each node as a processor and disk. The processor contains a CPU and a portion of local (main) memory available for caching. We chose an ethernet as an instance of the communications network. A sketch is shown in Figure 1. CPUs, disks and the communication network have associated priority queues which are serviced according to FIFO discipline. The centralized-server system consists of a single node that contains CPU, main memory, and the same number of disks as in the distributed database. CPU and disks are similarly serviced according to the FIFO discipline.

Figure 1: Distributed Main-Memory Database Architecture

2.2 The Memory Hierarchy

The memory hierarchy of the distributed main-memory database consists of local main memory, remote main memory (accessed over the communications network), and disk. In terms of access time, there are single order of magnitude differences between the local memory (tenths of a millisecond), remote memory (milliseconds), and disks (tens of milliseconds).

cost parameters	local disk read	local disk write	cache access	RPC overhead
cost values	50ms (disk)	55ms (disk)	1ms (CPU)	3ms (CPU) 1ms (net)

Table 1: Simulation Performance Parameters

In the simulation, an access cost is associated with each level of the memory hierarchy. These costs are listed in Table 1. If the communications network is slow, acting as a bottleneck, then there will be an even greater difference between local and remote storage access costs. This factor is included in simulation parameters, but we have not yet pursued it as a significant source of performance gains. We ignore CPU hardware caches completely. Their effects are local, extremely fine-grained and, being implemented in silicon, not subject to software solutions.

Within this memory hierarchy, our simulation manages a set of objects representing the contents of the database. At the disk storage level, the set of objects is completely partitioned: there is no disk-disk redundancy between sites. Memory-disk redundancy of objects is managed locally, in conjunction with the memory-memory redundancy as detailed below in section 2.5.

2.3 Object Location

There are three design decisions in a distributed database's cache management system: (1) the object location algorithm, (2) the replacement algorithm, and (3) the consistent update algorithm. In the simulation, we have adopted simple and realistic algorithms.

Object location algorithms find object copies in the system. Clearly, a centralized server does not need an object location algorithm: either the object is resident in main-memory or it is resident on disk. At the other extreme, the SDMD system allows the possibility of getting the object from *any* of the sites in the system. That is, even if the object is not found in a site's own cache, it may be resident in another site's main memory.

We use the following algorithm for object location in the SDMD architecture. If a site fails to find a copy in local cache, we use a broadcast request with individual replies and timeout. After queuing, each broadcast uses the network for half a millisecond. At every remote site, the object request interrupts any local transaction processing to emulate OS support and costs one cache access (one millisecond of CPU). All sites with a copy of the requested object in their caches queue for the network and reply: the requesting node simply discards all replies after the first. Expiration of a five-millisecond timeout period indicates that the object is not cached in any remote node; the object must then be fetched from remote disk. A table lookup determines which site has the object. In other words, when we need a copy of an object we traverse the memory hierarchy looking at local cache, remote cache, local disk and remote disk in turn.

The object location algorithm used in a distributed client-server system, is intermediate in complexity between the centralized-server and SDMD architectures. If a site does not have the datum in its cache, it can only request it from the owner-site. This (unique) site is determined by the same table lookup mechanism used by the SDMD.

2.4 Update Propagation

Consistent update algorithms keep all copies of an object mutually consistent. The Centralized-Server has (at most) one copy of an object to manage: update propagation is not complicated. In the distributed database, we use broadcast to propagate update values across the system. This takes half a millisecond on the network and a cache access for each node. This is a ''read-one, write-all'' strategy for maintaining replica control [1, 4]. In this simulation, in which we either read or write only one object at a time, timestamping the update messages suffices to maintain copy consistency.

2.5 Replacement Algorithm

Unbounded caching will eventually exhaust storage capacity. The criteria for choosing which copies reside at a given level of memory hierarchy are embodied in the replacement algorithms, so named for their similarity with the page replacement algorithms in virtual memory, such as Least Recently Used (LRU). The replacement algorithms are invoked when the storage capacity of a given level in the memory hierarchy (e.g. main memory) is full, to keep the most valuable objects and throw away the others. Replacement algorithms in a distributed environment can be complex, in the sense that an object's "value" can be a function of many parameters [6]. In this paper, the replacement algorithm used is Not Frequently Used with aging, i.e., each site approximates classical LRU behavior. We made a deliberate decision to define object value as a function of exclusively local parameters. Maintaining the accurate value of a global parameter is costly in a distributed system and even more so when the network scales up. The algorithm keeps objects with the highest local values in main memory and throws away those with low local values.

2.6 The Simulation Program

In our simulation, a central transaction server creates transactions with exponential inter-arrival times and distributes them uniformly over the nodes of the network (or the single centralized-server node). The probability of a transaction accessing a particular object is specified by a parameter that we term ''access locality''. This is similar to the notion of locality of reference in virtual memory in that access to objects is *not* distributed uniformly over the database. Access locality has the effect of creating a ''hot-set'' of accessed objects within the overall database. In our model of a hot-set we use an exponentially decaying curve, with a small set of objects receiving most of the accesses. Transactions access objects in either read or write mode. The mixture of read and write accesses is specified by a Bernoulli distribution.

In figure 2 we show the execution of a write transaction in the SDMD environment (the most complex of the systems presented here). The local node determines which part of the database holds the object on disk. A message is sent via the network to that processor to obtain a copy of the object. When the message arrives it is queued at the remote CPU. After processing by the CPU, the request is then queued at the remote disk. Once the object is returned from the remote disk it is sent over the network (queuing on the network queue again) to the local processor. The transaction can then execute at the local CPU: the modified object's value is propagated to all sites by an application of the consistent update algorithm. We optimize the simulation of CPU use by summing a transaction's CPU usage and then charging that usage as one block.

In figure 2 we also show the execution paths of a read transaction. A read transaction differs from a write in that it uses the full location algorithm and has no need to invoke the update propagation algorithm. The object location algorithm is invoked, checking local cache, remote caches and then disks in turn. In order to achieve the goal of fast performance, cache requests must be able to bypass executing transactions. This is done by assigning cache requests a higher priority than either disk or CPU processing jobs. As soon as the object is located, a copy is placed in local cache and used to execute the transaction. Costs are accounted in the same manner as write transactions.

Table 1 highlights the resource performance parameters while table 2 notes the remaining major parameters.

Figure 2: Transaction Overview

Parameter	Value(s)
Computing nodes	10
Networks	1
Objects	5000
Object Cache	0 to 1600 objects
Read/Write mix	read only to write only
Transaction cost	10ms CPU

Table 2: Model Parameters

3 Analysis of Results

Instead of throughput we used mean response time in comparing the performance gains for different transaction arrival rates. The batched-means method was used to ensure statistical robustness. We ran the simulations until the relative half-width of the confidence interval for the response time statistic, averaged over a minimum of ten batches, was 0.05 or less.

3.1 Remote Memory Performance Gains

The existence of a database hot-set implies that if sites can get most of the "hot" objects into cache, then performance will improve dramatically [7]. Figure 3 shows the mean response time of read-only transactions as a function of cache size. The curves represent different loads on the system, from transactions arriving every 6ms to every 20ms.[1] The mean response time decreases markedly with increased cache size. Since write transactions do not benefit from caching, with 50% of the transactions writing we have fewer benefits from caching, shown in the left of figure 3

Figure 3: All-Read and Half-Read Transactions with Caching

The critical difference between the SDMD and DCSD architectures is that in the symmetric architecture, sites are not restricted to requesting an object only from the owner site. Intuitively, even if the SDMD pays a higher communication cost because the network is used more intensively, there will be a net gain in performance because of the higher cache-hit ratio. In figure 4 we graph mean response time against increasing cache size (per node) for both the SDMD and DCSD architectures[2]. Clearly the symmetric architecture does much better than the client-server architecture (at some points, more than 50% better). To see why response time decreases so dramatically we graph the cache-hit ratios of the two architectures (against increasing cache size) to the right in figure 4. The solid line indicates the local site's hit ratio under both architectures, the dotted line is the hit ratio in the DCSD model, and the dashed line is the hit ratio in the SDMD model. In the DCSD system, an object can only be accessed at two sites (local and owner/server cache). When sites only have room for, say, 500 objects, then the overall hit-ratio is about 33% (versus 80% for SDMD). In contrast, sites in the SDMD system can access *any* other site (nine others, in this simulating), and the non-linear increase in hit-ratio reflects the greater number of available caches.

[1]Transaction interarrival rates are exponentially distributed. The range specified is that of the means.

[2]Transaction inter-arrival rate is 10ms, the read/write ratio is 1.0, and the other parameter values are as before

Figure 4: Performance of SDMD versus DCSD Architecture

3.2 Architecture Efficiency

In this section we investigate the *efficiency* of the symmetric architecture. By efficiency, we refer to the issue of how effectively are resources utilized in a multi-site SDMD as opposed to their utilization be a single, centralized manager. Because we are not attempting load balancing among the sites we cannot use one site's CPU resources to improve performance at another site. Memory, however, is, in a sense, distributed among the nodes-- total available memory is ten[3] times greater than that at any one node-- and we need to know whether all this memory is being put to good use. In the earlier graphs, for example, when sites have 500 units of memory the total available memory suffices to store the *entire* database in cache. In the SDMD architecture, this does not happen. Instead, each site stores as much of the hot-set in its own cache as it can: as a result, seldom accessed objects tend to remain on disk.

A centralized-server architecture will obviously manage memory more efficiently than SDMD because memory management is localized at one site. On the other hand, a single, more powerful processor may be more expensive than a few less powerful processors. In our simulation, response time is a function of CPU power (needed to process the CPU part of the transaction) and of available main memory (which reduces the need for disk I/O). System performance in a centralized-server architecture therefore trades off more efficient use of CPU and memory against (possibly) non-linear cost curves for CPU and memory. In order to get a feel for the efficiency of the SDMD architecture, we parameterized the centralized-server system as a single node that (1) has CPU_scale times the CPU power, and (2) has MEM_scale times the main memory of a of a single site in the SDMD. We then ran simulations to determine the CPU_scale and MEM_scale that enables the centralized_server to perform (approximately) as well as the SDMD. This tells us, in other words, how much more powerful a single, centralized site

---

[3]in the simulations

needs to be, as compared to a single site in a multi-site SDMD, in order to achieve the same performance.

Figure 5: Performance of SDMD versus Centralized Architecture (U = 15% and U = 30%)

In figures 5, and 6 we graph mean response time for SDMD and centralized-server models against increasing cache size. The three graphs differ in the average utilization of a single site in the SDMD (measuring, respectively, 15%, 30%, and 55%). Because the site must process other jobs besides transactions, these jobs divert CPU resources from the transactions. As a result, as site utilization increases, the centralized-server needs greater amounts of CPU power and main memory in order to achieve the performance of the SDMD system. We see that when a single site in the SDMD is 55% utilized, the centralized-server must have five and a quarter times the CPU power, and four times the memory in order to perform as well as the aggregate SDMD system. Depending on the cost functions of CPU and memory, one architecture will make "more sense" than the other in terms of price/performance ratios.

The fact that the centralized-server needs only four times the memory (in figure 6) indicates that memory is not used as effectively as possible in the SDMD system. This is corroborated by a study of object redundancy in SDMD which showed that *all* sites will tend to maintain copies of "hot" objects in their cache, instead of relying on a copy available in other sites. As a result, less hot objects are crowded out of cache (instead of replacing a highly available hot object) with a consequent degradation of performance. This behavior points to the need for more sophisticated memory management algorithms in the SDMD environment that will (1) be efficient, and (2) not violate site autonomy [6, 7].

Figure 6: Performance of SDMD versus Centralized Architecture (U = 55%)

4 Conclusion

We have described a symmetric distributed main-memory database architecture and studied the effects of caching on performance. The salient feature of the architecture is node symmetry, because nodes provide each other data from their main memory through a fast network. The architecture captures today's environment where networks connect many powerful workstations. Even though we know that simple algorithms suffice for simple memory hierarchies [2], the general symmetric architecture will require better caching and replication management to realize the potential for higher performance and availability.

Simulations were used to study the performance gains in a symmetric distributed main-memory database (which makes use of other site's memory) as compared to a distributed client-server architecture (where sites do not take as much advantage of the distributed resources). For a wide range of cache sizes and object access patterns, we find the hit ratio of remote memory cache to be several times the hit ratio of local cache. This result demonstrates the importance of remote memory cache in the symmetric architecture, due to both its size and speed. Since the size of remote memory grows larger as the network grows and remote memory access becomes faster as the network evolves, remote memory cache will become ever more important.

We also studied the efficiency of the SDMD architecture, in the sense of determining how effectively CPU and memory resources are used in a multi-site system as compared to a single, more powerful, centralized-server. Even though SDMD performs better than a distributed client/server architecture (because it makes better use of system redundancy), we found that "too much" cache redundancy exists in the SDMD system, pointing to the need for more sophisticated cache management strategies in this environment. Depending on the cost of CPU and memory, a SDMD architecture may still perform better than a centralized system.

References

1. P.A. Bernstein and N.Goodman. "An algorithm for concurrency control and recovery in replicated distributed databases". *ACM Transactions on Database Systems 9(4):596--615*, (December 1984).

2. W.J. Bolosky, R.P. Fitzgerald, and M.L. Scott. "Simple but effective techniques for memory management". *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles pages 19-31*, (December 1989).

3. M. Schroeder and M. Burrows. "Performance of Firefly". *Proceedings of the Twelfth Symposium on Operating Systems Principles ACM/SIGOPS*, (December 1989).

4. C. Pu, J.D. Noe, and A. Proudfoot. "Regeneration of replicated objects: A technique and itsimplementation". *IEEE Transactions on Software Engineering SE-14(7):936--945*, (July 1988).

5. M. H. MacDougall. *Simulating Computer Systems.* Computer Systems MIT Press, 1987.

6. Pu, C., Leff, A., Korz, F., and Chen, S-W. Valued Redundancy. Tech. Rept. CUCS-453-89, Columbia University, 1989.

7. Pu, C., Leff, A., Korz, F., and Chen, S-W. Redundancy Management in a Symmetric Distributed Main-Memory Database. Tech. Rept. CUCS-014-090, Columbia University, 1990.

Table of Contents

1 Introduction	**0**
2 Simulation Study	**1**
2.1 The Architectures	**1**
2.2 The Memory Hierarchy	**2**
2.3 Object Location	**3**
2.4 Update Propagation	**3**
2.5 Replacement Algorithm	**4**
2.6 The Simulation Program	**4**
3 Analysis of Results	**5**
3.1 Remote Memory Performance Gains	**5**
3.2 Architecture Efficiency	**7**
4 Conclusion	**9**

List of Figures

Figure 1:	**Distributed Main-Memory Database Architecture**	**2**
Figure 2:	**Transaction Overview**	**5**
Figure 3:	**All-Read and Half-Read Transactions with Caching**	**6**
Figure 4:	**Performance of SDMD versus DCSD Architecture**	**7**
Figure 5:	**Performance of SDMD versus Centralized Architecture (U = 15% and U = 30%)**	**8**
Figure 6:	**Performance of SDMD versus Centralized Architecture (U = 55%)**	**9**