Centralized versus Distributed Index Schemes in OODBMS — A Performance Analysis

Julie Basu
julie@cs.stanford.edu
Stanford University
Computer Science Department
and Oracle Corporation

Arthur M. Keller
ark@cs.stanford.edu
Stanford University
Computer Science Department
Stanford, CA 94305-9020, USA

Meikel Pöss
s_poess@ira.uka.de
Technical University of Berlin
Computer Science Department
D-10587 Berlin, Germany

Abstract

Recent work on client-server *data-shipping* OODBs has demonstrated the usefulness of local data caching at client sites. In addition to data caching, *index* caching can provide substantial benefits through associative access to cached objects. Indexes usually have high contention, and database performance is quite sensitive to the index management scheme. This paper examines the effects of two index caching schemes, one centralized and the other distributed, for index page management in a *page server* OODB. In the centralized scheme, index pages are not allowed to be cached at client sites and are managed by the server. The distributed index management scheme supports inter-transaction caching of index pages at client sites, and enforces a distributed index consistency control protocol similar to that of data pages. We study via simulation the performance of these two index management schemes under several different workloads and contention profiles, and identify scenarios where each of the two schemes performs better than the other.

1 Introduction

In recent years much research has been done on object-oriented database management systems (OODBMS). Many DB vendors now offer commercial systems, and these systems have gained true acceptance for certain classes of applications, e.g., CAD/CAM and CASE. OODBMS typically have client-server architectures, and commonly use the *data-shipping* approach [4] to move data from the central database server to local client caches. The data granularity can vary from objects to pages. Page-level data shipping has been found to have superior performance under many different load conditions over *object servers*, which use individual objects as *logical* units of data transfer [3]. The data-shipping strategy allows query processing to occur locally at client sites, allowing efficient fine-grained interaction between the application and the DBMS. It exploits the resources of client workstations, leading to better scalability and performance.

Despite the potential cost of maintaining the consistency of local caches, many studies have demonstrated the general positive effects of client-side data caching on system performance [2, 3, 6, 7, 14, 15]. Local caching and inter-transaction reuse of pages can reduce network traffic, offload the server, and minimize query response times.

1.1 Navigational versus Associative Access to a Client Cache

A major assumption in the above studies has been that the transactions access the client cache *navigationally*, i.e., only through object IDs (OIDs). For example, an input transaction is represented by a list of object references in [7]. This model is inadequate because it ignores the commonly-used *associative* queries and updates, which specify a target set of objects using general predicates on one or more attributes of an object class.

In order to support associative queries at client sites, we need *content-based* access to the cached data, i.e., access based on the values of the object attributes and not merely on their IDs. During query planning and execution phases, indexes defined at the server are used to provide efficient associative access to the central database. One way of providing associative access to a client cache is to allow caching of the server index pages at the client site, so that objects satisfying an index-based associative query can be identified locally. However, index pages generally contain many more entries compared to a data page, often making them the most heavily used *hot spots* in a database. The system response is therefore sensitive to the number and frequency of index access and updates. The tradeoff between the different performance factors involved in caching of server index pages thus deserves detailed study.

1.2 Issues and Trade-Offs in Index Caching

Index pages differ from data pages in the following key respects:

- **Index page entry size:** The size of an index page entry is normally much smaller than the size of an object on a data page. This leads to a dense aggregation of entries making the sharing and contention on index pages to be the highest in the database.
- **Page reference pattern for index writes:** Index pages have to be scanned in order to determine which objects satisfy an associative query. Modification of an indexed attribute of a data object will cause one or more index pages to be updated. The index page referencing pattern is thus quite different from that of data pages.

- **Index write probability:** The index write probability per data object update is a different quantity and should be considered separately.

For index-based associative queries, local caching of indexes allows a client to determine if some or all of the desired objects are available in its cache. Cached objects need not re-fetched from the remote server, saving network traffic and improving query response times. If the caching policy disallows storage and reuse of index pages at client sites, then all index page references for range queries and index updates at the client must be routed to the server, which may become a bottleneck in the system. However, local index caching requires the enforcement of a distributed index consistency control protocol, which may be expensive in certain update-intensive scenarios.

1.3 Our Focus

In this paper, we investigate the performance of a page server OODBMS supporting not only navigational, but also associative client cache access via index page caching. We extend the simulation model of [3] to incorporate index-based associative queries in transactions, consider index read and update costs for query processing, and quantify the benefits and drawbacks of index page caching at client sites. We contrast two different schemes: (1) a *centralized* scheme that maintains all index pages at the server with no client-side index caching within or across transactions, and (2) a *distributed* index caching scheme in which inter-transaction reuse of index pages is supported by a distributed consistency control protocol. In both cases, data pages are cached on clients across transactions. We develop simulators for both the schemes, and experimentally analyze the system behavior under several different workload patterns.

2 Related Work

The performance benefits of client-side data caching schemes for OODBMSs have been investigated in several recent studies [2, 3, 6, 7, 14, 15]. These papers consider caching of pages in general without examining index pages in particular. For example, in the simulation study of [3], the only way to represent indexes is in the form of shared "hot" regions with high contention. As noted in the Introduction, we believe this is not an adequate model of index behavior; access, update and contention characteristics of index pages are very different from those of data pages, and in particular from the data page read and write patterns considered in [3].

Index partitioning among different sites in the context of distributed databases has been investigated in [11]. Their focus has also been on range queries. However, in contrast to our dynamic caching environment, the partitioning in [11] is static and does not vary with query patterns at a site.

In a broader perspective, the issue of supporting associative access to a client-side cache has been addressed in [13] and [10] — the first applies *ViewCache* techniques with local indexes on the client cache in the ADMS± system, while the second examines a *predicate-based* caching scheme. Another paper [5] presents a *semantic* caching approach. Unlike these studies, the work reported in this paper uses server indexes to support associative access to the client cache.

3 Centralized and Distributed Schemes for Index Management

We consider two different schemes, one centralized and the other distributed, for client-side caching of server index pages. In both the schemes, we use the PS-AA method of locking and replica control for data pages [3]. This method provides *adaptive* granularity for concurrency control and consistency maintenance of cached data, while using fixed-size pages as the data transfer unit. The PS-AA method switches from page-based to object-based locking when finer-grained sharing is deemed better, and uses *callbacks* for coordinating updates. For a variety of workloads studied in [3], the adaptive page server following PS-AA caching strategy was found to have consistently good performance, generally outperforming the other static and dynamic caching strategies investigated.

In the *centralized* index management scheme, all index pages are centrally stored and managed exclusively by the server, which treats them very differently from data pages. Query execution occurs in two steps. First, the client submits an associative query to the server, which sends back only the object IDs satisfying the query. The client then checks these object IDs against its cached objects and fetches only the missing objects from the server.

In the *distributed* index caching scheme, any client may cache index pages locally. Whenever an index page has to be read for an associative query, the client requests the server for a copy only if it is not available locally. Thus, cached index pages have implicit permission for local read. However, updating a cached index page requires coordination through the server. For index writes, we follow a policy similar to the *Callback-Read* [6] scheme for data pages, with the difference that a copy of the updated index page is sent to the server immediately after the update, allowing any other clients waiting to read or write the same index page to proceed.

4 Simulation Model

In this paper, we provide a brief description of our simulation model; further details may be found in [1]. The basic page server model is the same as that of [3]. We have reproduced in Figure 1 the page server simulation model from [3], with an additional "index manager" module on the server side for our index extensions. The clients — for brevity we show only one — are connected via a network to the server. We describe here only the index handling; details on the general page server scheme may be found in [3].

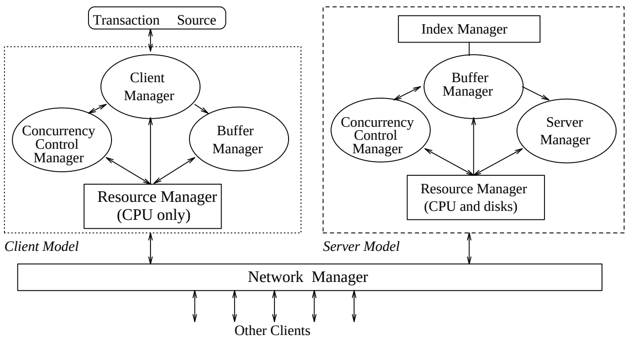


Figure 1. Simulation Model of a Page Server OODBMS

4.1 Modeling Indexes

Each associative access is modeled as an index read. Multiple index pages could be read depending on the size of the query. The result of an index range scan is a list of OIDs that is sent to the client which fetches objects as necessary, with object and index writes occurring according to the specified workload. For each index range read, the centralized scheme results in a round-trip to the server. Network and index lookup costs are associated with this operation. The distributed scheme incurs a similar overhead only if an index page is not locally available.

Each modification of a data object may cause one or more index updates based on the index write probabilities (variable parameters in our simulation). In our centralized scheme, indexes update requests are sent to the server. An index update for the distributed scheme requires communication with the server to obtain write permission. The updated index page is sent to the server immediately, but local data modifications are communicated only when a transaction commits [3]. Since appropriate locks are held on the data objects themselves, updating the index before a transaction commits cannot cause erroneous behavior — if any transaction reads an uncommitted index entry, it will subsequently block upon a read or write request for the associated data object until the updating transaction terminates. All index writes are undone upon transaction aborts, which may be caused by the detection of deadlocks.

Our simulator models two server indexes, one clustered and the other unclustered, named *cix* and *uix* respectively. It is important to consider both types of indexes, since the index usage, and data access and update patterns are different for the two. A query that uses a range predicate on an attribute with a clustered index will retrieve a set of objects grouped closely together in some set of data pages. On the other hand, access through an unclustered index will result in random data pages being fetched in. Clustered indexes are however less likely to be updated, because data objects are usually clustered in pages based on an attribute that is infrequently modified, such as the department number of employee tuples in an employee relation.

During the initialization phase of the simulator, both types of index pages are explicitly populated with the associated object IDs. Clustered index pages have OIDs in the sequential order of data clustering, while the unclustered index is randomly ordered. To avoid aberrations in the results due to a cold start with empty caches, both the server and the client caches are pre-loaded with pages. The server buffer manager first loads in all index pages during initialization, since index pages are frequently referenced. The remaining server buffer space is filled with data pages selected randomly. In contrast, only data pages and no index pages are cached by the clients at the start of simulation.

4.2 System and Cost Parameters

General system and cost parameters and their values for the simulation experiments are listed in Table I – these are identical to those assumed in [3]. The simulator is written in the Modula-2 based simulation language DeNet [12], and uses the same code as in [3] for the basic PS-AA page server. Thus, we have the exact behavior for data page caching as reported in [3], with index page reads, writes and local index caching being our newly added functionality.

An index entry is assumed to be 16 bytes, with 8 bytes each for the attribute value and the associated OID. For our page size of 4 Kbytes, this implies 250 object entries in an index page, as opposed to 20 objects per data page. Our database contains 25,000 objects in total, and therefore the clustered and unclustered indexes occupy 100 pages each. We assume a cost of 1000 machine instructions to locate the leaf index page holding the first object entry in the read interval. Once this entry is located, each consecutive index entry is read at a cost of 10 instructions, until the end of the read interval is reached. Index page updates also require the lookup cost, along with a write latch acquisition cost of 50 instructions and an update cost of 2000 instructions. We assume that new index entries are placed randomly in the index pages without overflow or underflow.

Table I: General System and Cost parameters

Parameter	Meaning	Value
ClientCPU	Instruction rate of client CPU	15 MIPS
ServerCPU	Instruction rate of server CPU	30 MIPS
NumClients	Number of client workstations	10
ClientBufSize	Per-client buffer size	25% of DBsize
ServerBufSize	Server buffer size	50% of DBsize
ServerDisks	Number of disks at server	2 disks
MinDiskTime	Minimum disk access time	10 milliseconds
MaxDiskTime	Maximum disk access time	30 milliseconds
Network-Bandwidth	Speed of network communication	80 Mbits per second
PageSize	Size of a page (data transfer unit)	4096 (4K) bytes
DBsize	Size of the database	1250 pages (5 MB)
ObjsPer-DataPage	Number of objects per data page	20 objects
FixedMsgCost	Fixed instructions per message	20,000 instructions
PerByte-MsgCost	Additional instructions per message byte	10,000 per 4KB page
Control-MsgSize	Size of a control message	256 bytes
LockCost	Cost per lock/unlock pair	300 instructions
Register-CopyCost	Cost to register/uregister a page copy	300 instructions
DiskCost	Cost of performing a disk I/O	5000 instructions
ReadObjCost	Mean cost to read an object	5000 instructions
WriteObjCost	Mean cost to write an object	10000 instructions
DataPage-MergeCost	Cost to merge two copies of a data page	300 instructions per object

4.3 Workload Model

Our workload model consists of two transaction types: *associative* and *navigational*. The former accesses data using an index; the latter is a list of object references by OIDs, as in [3]. For the purposes of our simulation, an associative transaction consists of a single range query or update, which is expressed in terms of a linear range on either the clustered or the unclustered index.

Client-specific workload profiles and general transaction parameters are summarized in Table II(a) and II(b). These parameters vary by the choice of workload type — HOTCOLD, UNIFORM, or HICON, which are very similar to the workloads used in [3].

The HOTCOLD workload has a high degree of access locality per client and a moderate amount of shared data contention. For the UNIFORM workload, data object references are assumed to be uniformly random over all the 1250 data pages. The HICON workload is a skewed workload representing high data contention amongst the clients — a small part of the database is accessed by all clients very frequently. Both clustered and unclustered indexes are

Table II(a): Workload Profile for Client i, $i = 1..10$

Parameter	Meaning
HotDataPgs	Hot range of data pages
ColdDataPgs	Cold range of data pages
HotCixPgs	Hot range of clustered index pages
ColdCixPgs	Cold range of clustered index pages
HotUixPgs	Hot range of unclustered index pages
ColdUixPgs	Cold range of unclustered index pages
AccHotDataProb	Probability of accessing a hot data page
AccHotCixProb	Probability of accessing a hot *cix* page
AccColdCixProb	Probability of accessing a cold *cix* page
AccColdUixProb	Probability of accessing a cold *uix* page

Table II(b): Workload Profile for Client i, $i = 1..10$

	Workload Type		
Parameter	HOTCOLD	UNIFORM	HICON
HotDataPgs	h to $h+124$, $h = 125*(i-1)+1$	—	1 to 250
ColdDataPgs	rest of DB	whole DB	rest of DB
HotCixPgs	c to $c+9$, $c = 10*(i-1)+1$	—	1 to 20
ColdCixPgs	rest of *cix* pgs	all *cix* pages	rest of *cix* pgs
HotUixPgs	—	—	—
ColdUixPgs	—	all *uix* pgs	—
AccHotDataProb	0.8	—	0.8
AccHotCixProb	0.8	—	0.8
AccColdCixProb	0.2	uniform	0.2
AccCold-UixProb	—	uniform	—

considered for associative access to data.

Transaction characteristics for all workload types are listed in Table III. The set of first six parameters is kept fixed while the second set of parameters is varied in our experiments. The probability that a transaction is associative and accesses data via an index is represented by the *AssocProb* parameter. A value of 0 for *AssocProb* implies that none of the transactions use either of the two indexes for accessing data; although there are no index reads in this case, index writes may occur upon data object updates. Given an associative transaction, *CixAccProb* denotes the probability that data is accessed via the clustered *cix* index.

An important characteristic of a transaction is its *size* or length, which is the number of data objects it accesses. In our model, there are four parameters that relate to the size of a transaction — *NumPages*, *PageLocality*, *CixReadSize*, and *UixReadSize*. *NumPages* denotes the mean number of pages accessed by a navigational transaction. The size of a navigational transaction in terms of the number of data objects it accesses is determined by the *PageLocality* parameter, which has a range of 1–7 with a mean of 4. In contrast, the size of an associative transaction is controlled by the *CixReadSize* and *UixReadSize* parameters. If data is accessed via the clustered index, *CixReadSize* defines the mean number of index entries scanned by the range query, and thus also the size of the transaction. *UixReadSize* is the corresponding parameter for the unclustered index.

A read-write transaction chooses to update a data object with probability *ObjWrtProb*. A data object write may result in updates to one or both the indexes, depending on the attributes updated. *CixWrtProb* and *UixWrtProb* respectively denote the probability of a clustered or an unclustered index update upon a data object write. These parameters have fixed values for our experiments, 10% and 80% respectively. To accurately model the probability that a new index entry is in a page different from the old one, we added the

Table III: Transaction Parameters for all Workloads

Parameter	Meaning	Value
ThinkTime	Mean think time between transactions	0
PageLocality	No. of objects accessed per page by a navigational transaction	1-7 (min-max), mean 4
CixWrtProb	Probability of clustered index update per object write	0.1
UixWrtProb	Probability of unclustered index update per object write	0.8
Cix2pgWrtPb	Probability that a clustered index update modifies two index pages	0.8
Uix2PgWrtPb	Probability that an unclustered index update modifies two index pages	0.8
ReadOnlyProb	Probability that a transaction is read-only	varies
AssocProb	Probability that a transaction is associative	varies
CixAccProb	Probability that an associative transaction accesses data via the clustered index	varies
UixAccProb	Probability that an associative transaction accesses data via the unclustered index	$1 - CixAccProb$
NumPages	Mean no. of data pages accessed per navigational transaction	varies
CixReadSize	Mean no. of objects in a range scan of clustered index	varies
UixReadSize	Mean no. of objects in a range scan of unclustered index	varies
ObjWrtProb	Data Object Write Probability	varies

two parameters *Cix2PgWrtPb* (clustered) and *Uix2PgWrtPb* (unclustered). We do not consider object deletes and inserts in this study, although they could be incorporated in our simulation model.

There are two different parameters controlling object writes — namely, *ObjWrtProb* and *ReadOnlyProb*. The former parameter controls object writes on a per-transaction granularity, while the latter is a per-object write probability for a given read-write transaction. A read-only transaction occurs with probability *ReadOnlyProb*, and has no object writes. Therefore, the *ObjWrtProb* parameter is not relevant for a transaction that is read-only. Both object reads and updates are performed by a read-write transaction, which occurs with probability $(1 - ReadOnlyProb)$, with *ObjWrtProb* being the probability of an object accessed by a transaction is written by it.

5 Simulation Results

To validate our simulator, we first verified that in the absence of associative queries (i.e., index reads) and index updates, both the centralized and distributed index algorithms yield exactly the same results. These results are also in agreement with the results reported in [3]. Experiments were performed to evaluate the effects of the centralized and distributed index schemes under different load and access patterns. In the results reported below, we focus primarily on associative transactions, and on index behavior with high write probability. Throughput in terms of number of transactions per second (TPS) is our main performance metric, but other metrics such as the number of remote index read and write requests are also tracked. We use these two measures in particular to analyze the results of our experiments.

Due to space constraints, we discuss below some representative cases in which the two major write parameters in our model, namely, *ReadOnlyProb* and *ObjWrtProb*, are varied. The general trend we observed was that in the presence of update (read/write) transactions, there is significant deterioration in performance for the distributed index scheme, even with quite low object write probabilities. This behavior is not surprising given the cost associated with acquiring a distributed latch on an index page. However, as demonstrated in Figures 1(a) through 1(d), the ratio of update versus read-only transactions and the workload type are important factors in determining whether the centralized index scheme performs better than the distributed or vice-versa.

Figures 1(a) and 1(b) show the effect of varying the ratio of read-only versus update transactions for the HOTCOLD workload, using associative transactions of size 30 objects and the clustered index. The object write probability is 10%. The cross-over in throughput occurs when about 75% of the transactions are read-only, beyond which point the performance of the distributed index scheme increases rapidly, exceeding the much slower growth for the centralized. The number of remote index read requests per transaction, as plotted in Figure 1(b), shows a rapid decrease for the distributed scheme with increasing number of read-only transactions, falling below that of the centralized when update transactions are less than 15% of the transaction load. Similar behavior is observed in the case of the HICON workload, with the cross-over in throughput occurring at a mix of 70% read-only queries and 30% update transactions (Figure 1(d)).

Figure 1(a): Read-Write Clustered With Varying Update Transactions

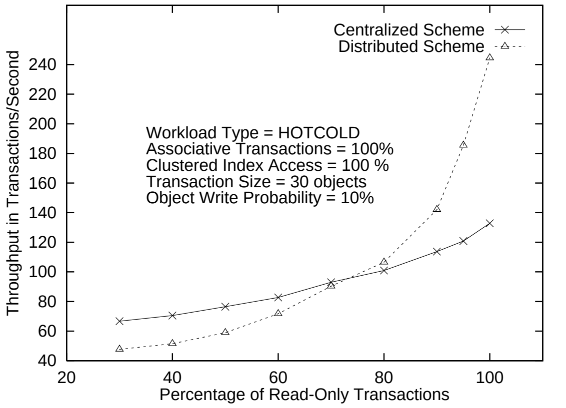
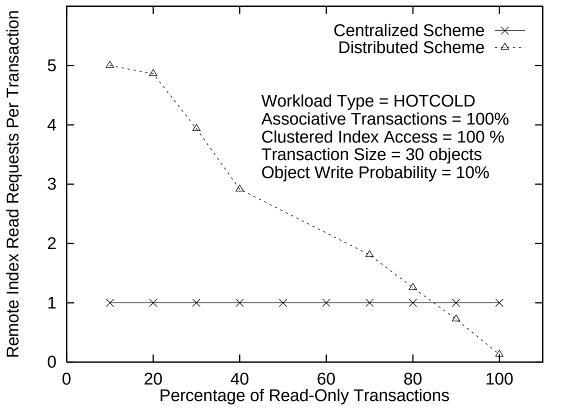


Figure 1(b): Read-Write Clustered With Varying Update Transactions



Figures 1(c): Read-Write Clustered with Varying Update Transactions

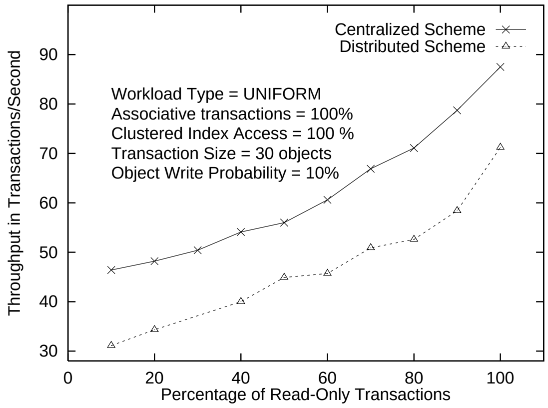


Figure 1(d): Read-Write Clustered with Varying Update Transactions

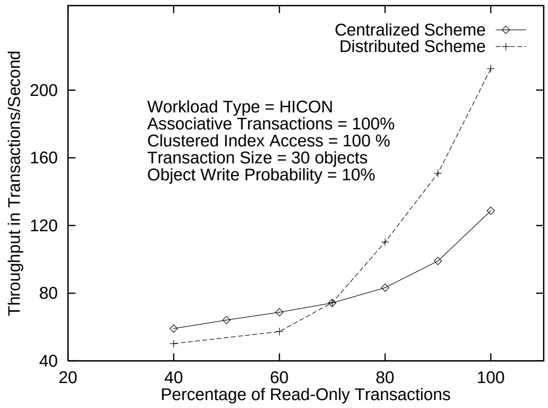
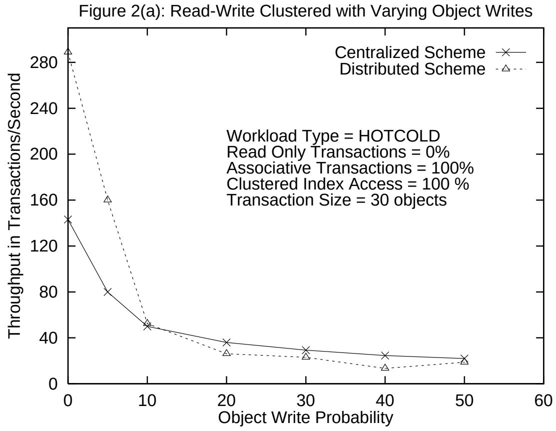


Figure 2(a): Read-Write Clustered with Varying Object Writes



The performance characteristics of the UNIFORM workload under the same conditions are quite different from the HOTCOLD and HICON cases, as illustrated in Figure 1(c). Uniformly random access to all the 100 pages of the *cix* index causes a larger number of index pages to be cached in the client buffer, reducing the data page hits and the throughput. Therefore, the distributed index scheme never beats the centralized one, even with a 100% read-only load.

Figures 2(a) and 2(b) report the throughput of the HOTCOLD and HICON workloads with varying probability of object writes. For low values (less than 10%) of object update probability, the distributed index performs better than the centralized one with the given parameter settings. The performance of the UNIFORM workload (not shown here due to space restrictions) is different in that the centralized scheme is better than the distributed one for all object write probabilities.

Figure 2(b): Read-Write Clustered with Varying Object Writes

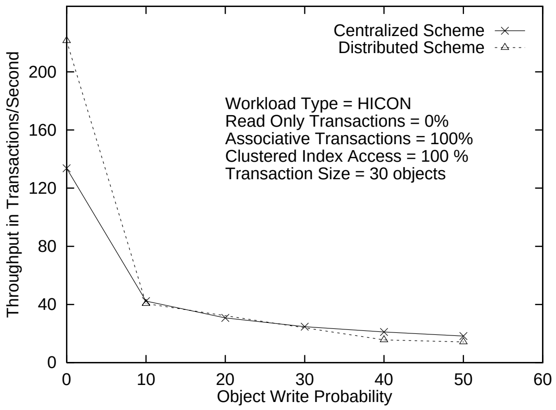


Figure 2(c): Read-Write Navigational with Varying Object Writes

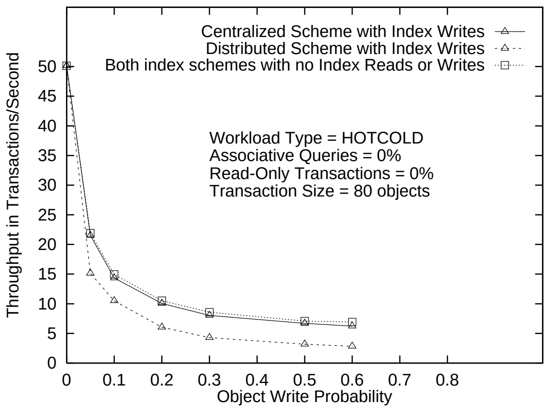


Figure 2(d): Read-Write Clustered with Varying Transaction Size

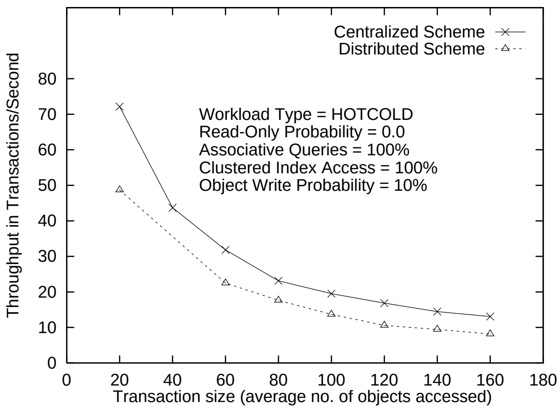


Figure 2(c) above shows the system performance in terms of throughput for a load of exclusively navigational queries and mean transaction size of 80 objects. Notice that the centralized scheme closely follows the curve of no index reads or write. However, the distributed scheme with the same parameters lags behind, even for small values (5%) of the data object write probability.

In Figure 2(d), the object write probability is 10% and all transactions are read-write. The performance of the distributed index scheme trails the centralized one for all transaction sizes between 20 through 160 objects.

Figure 3(a): Read-Only Clustered with Varying Associative Access

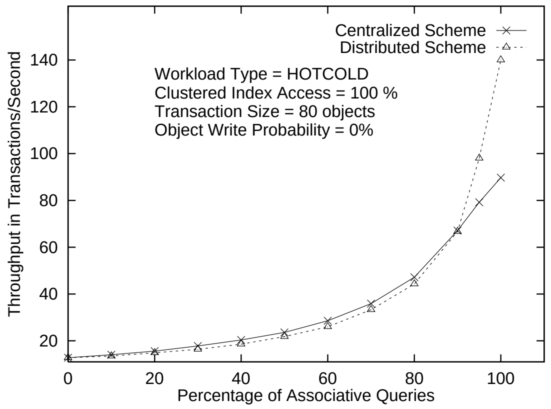
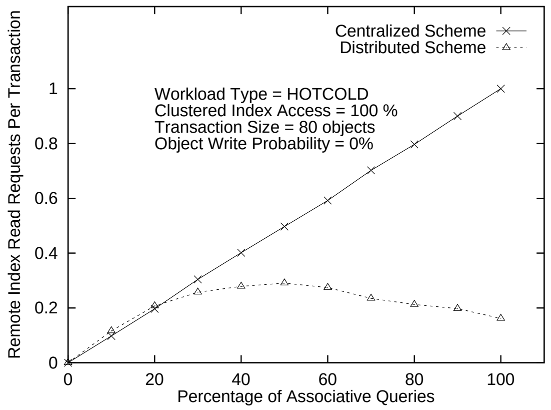


Figure 3(b): Read-Only Clustered with Varying Associative Access



Finally, we discuss some results of the read-only case. Figure 3(a) through 3(d) demonstrate the effect on the throughput by varying the ratio between associative and navigational access. The transaction size is kept constant at 80 objects, and associative data access is entirely through the clustered index. As shown in Figure 3(a) and 3(c), varying the ratio of associative versus navigational access causes the throughput to vary substantially for both workload types (HOTCOLD,HICON). The distributed scheme starts out as slightly worse than the centralized scheme, but does better beyond 85% in the HOTCOLD case, and beyond 90% in the HICON case using associative access. For both workloads, medium and high values of associative clustered access generate a lower number of index page read requests at the server for the distributed scheme than in the centralized, indicating that index page caching is indeed effective in increasing the throughput by causing index hits for associative queries.

Figure 3(c): Read-Only Clustered with Varying Associative Access

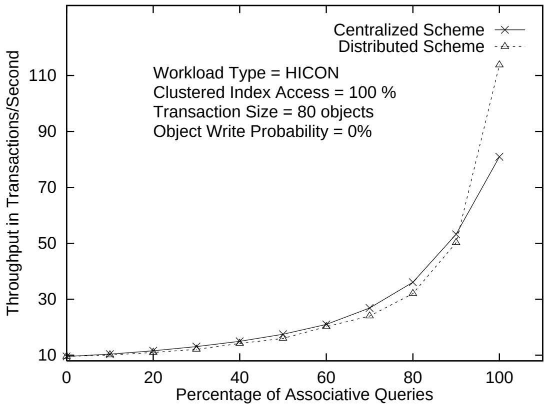
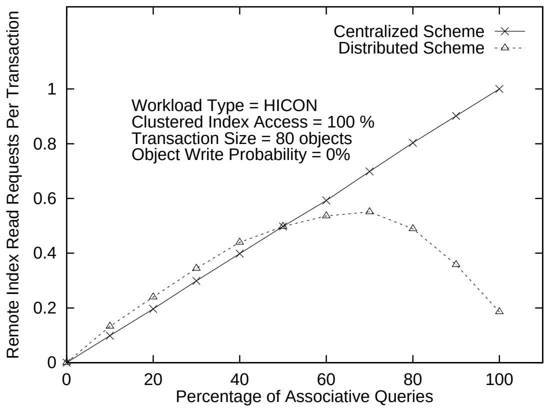


Figure 3(d): Read-Only Clustered with Varying Associative Access



6 Conclusion

Indexes can provide efficient associative access to data, and their access and maintenance are important performance considerations for a database system. We have extended the page server simulation model of [3] to incorporate clustered and unclustered indexes and associative (range) queries, and have evaluated through detailed simulation experiments the performance of two different schemes, one centralized and the other distributed, for index management in a page server OODB. The former executes all index read and write operations at the server site, while the latter allows index page caching and reuse at client sites following a distributed consistency control protocol. We have considered three separate workload types, HOTCOLD, UNIFORM, and HICON, modeling different data and index usage patterns at the clients, and have analyzed performance of the two index schemes under various parameter settings.

Reviewing our results, we find that for the UNIFORM workload, the centralized index always performs better than or is comparable to the distributed scheme. This result, although rather surprising for the read-only case, is due to the fact that index access is uniform in this workload, leading to a larger number of references to different index pages and their caching in the client buffer. The effective buffer space for data pages at the client site is thus reduced, and so is the system throughput.

Table IV: Index Performance with Different Workloads, Transaction size 30 objects

Workload Type	Read-Only Transactions	Read-Write Transactions with 10% Object Write Probability	
		< 75% Read-Only	> 75% Read-Only
UNIFORM, > 70% clustered, all associative	Centralized better	Centralized better	Centralized better
UNIFORM, < 70% clustered, all associative	Both schemes comparable	Centralized better	Centralized better
HOTCOLD/ HICON, > 90% associative, all clustered	Distributed better	Centralized better	Distributed better
HOTCOLD/ HICON, < 90% associative, all clustered	Both schemes comparable	Centralized better	Both schemes comparable

The picture for HOTCOLD and HICON cases is quite different in the presence of many updates; centralized indexes in general perform better than distributed, even for rather small write probability (10%) of data objects affecting an index. This behavior is a result of the high cost involved in obtaining exclusive distributed write latches on index pages. The gap narrows, however, as the percentage of read-only transactions increases. Cross-over points in throughput were obtained in our experiments with a varying mixture of read-only and read-write transactions, with the distributed scheme overtaking the centralized for high occurrences (about 70% or more) of read-only queries. As the object write probability increases, this crossover point migrates to require even more read-only transactions for the distributed scheme to dominate. We suggest that centralized indexes be used whenever there are a significant percentage of transactions updating indexes; otherwise, distributed index are better, at the cost of some architectural and implementation complexity.

We summarize in Table IV the behavior of the centralized and index schemes for different workload types. We present also the results for read-only transactions, which have not presented in detail because of space constraints. The results hold for a transaction size of 30 objects and an object write probability of 10%. The behavior is quite similar for larger transaction sizes; however, increasing the object write probability reduces the performance of the distributed scheme.

In conclusion, indexes behave very different than data in a page server OODBMS. While client-based query processing appears profitable for data pages, centralized (i.e., server-based) index processing seems superior for some workloads. In light of these results, we believe a "hybrid" approach of both server and client-based query processing might be the best alternative for a page server OODB, as discussed in [8]. Further work remains to be done in this area to clearly identify the performance trade-offs, taking into account both data and index usage patterns. A relaxed index consistency algorithm is proposed in [9]; such special algorithms can improve the performance of our distributed index caching scheme. Using more advanced techniques for index management are areas of future work.

Acknowledgements

We are very grateful to Prof. Michael Carey for providing us the source code of the simulator developed for the work reported in

[3], and to Markos Zaharioudakis for clarifying several questions on the implementation of the simulator. Their help has greatly simplified our simulation task. We also thank Prof. Gio Wiederhold for his guidance and support. The work of Julie Basu was supported in part by Oracle Corporation, and by an equipment grant from Digital Equipment Corporation.

References

[1] J. Basu, M. Poess, and A. M. Keller, Performance Evaluation of Centralized and Distributed Index Schemes for a Page Server OODBMS," *Technical Report No. STAN-CS-TN-97-55*, Computer Science Department, Stanford University, March 1997.

[2] M. Carey, M. Franklin, M. Livny, and E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architecture," *ACM SIGMOD Intl. Conf. on Management of Data*, Denver, CO, May 1991, pp. 357-366.

[3] M. Carey, M.J. Franklin, and M. Zaharioudakis, "Fine-Grained Sharing in a Page Server OODBMS," *ACM SIGMOD Intl. Conf. on Management of Data*, Minneapolis, MI, May 1994, pp. 359-370.

[4] D. J. DeWitt, D. Maier, P. Futtersack, and F. Velez, "A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems," *16th Intl. Conf. on Very Large Data Bases*, Brisbane, Australia, 1990, pp. 107-121.

[5] S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava, and M. Tan, "Semantic Data Caching and Replacement," *22nd Intl. Conference on Very Large Data Bases (VLDB 96)*, Bombay, India, September, 1996, to appear.

[6] M.J. Franklin, "Caching and Memory Management in Client-Server Database Systems," *PhD thesis*, University of Wisconsin-Madison, 1993.

[7] M.J. Franklin, M.J. Carey, and M. Livny, "Local Disk Caching for Client-Server Database Systems," *19th Intl. Conf. on Very Large Data Bases*, Dublin, Ireland, August 1993, pp. 641-654.

[8] M.J. Franklin, B.T. Jonsson, and D. Kossmann, "Performance Tradeoffs for Client-Server Query Processing," *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 1996, pp. 149-160.

[9] V. Gottemukkala, E. Omiecinski, and U. Ramachandran, "Relaxed Index Consistency for a Client-Server Database," *Intl. Conf. on Data Engineering, 1996*, Birmingham, U.K., pp. 352-361.

[10] A.M. Keller and J. Basu, "A Predicate-based Caching Scheme for Client-Server Database Architectures," *The VLDB Journal,* Vol. 5, No. 1, Jan 1996, pp. 35-47.

[11] J. Liebeherr, E.R. Omiecinski, and I.F. Akyildiz, "The effect of index partitioning schemes on the performance of distributed query processing," *IEEE Transactions on Knowledge and Data Engineering*, June 1993, vol.5, no.3, pp. 510-522.

[12] M. Livny, "DeNet User's Guide (Version 1.5)," Computer Sciences Department, University of Wisconsin-Madison, 1990.

[13] A. Delis and N. Roussopoulos, "Performance and Scalability of Client-Server Database Architectures," *18th Intl. Conf. on Very Large Data Bases*, Vancouver, B.C., Canada, 1992, pp. 610-623.

[14] Y. Wang and L.A. Rowe, "Cache Consistency and Concurrency Control in a Client-Server DBMS Architecture," *ACM SIGMOD Intl. Conf. on Management of Data*, Denver, CO, May 1991, pp. 367-376.

[15] K. Wilkinson and M.-A. Neimat, "Maintaining Consistency of Client-Cached Data," *16th Intl. Conf. on Very Large Data Bases*, Brisbane, Australia, 1990, pp. 122-133.