A shorter version appears in *Proc. Int. Conf. Computers and Their Applications*, Tempe, Arizona, March 1997.

Performance Evaluation of a Two-Level Hierarchical Parallel Database System

Yifeng Xu and Sivarama P. Dandamudi
School of Computer Science, Carleton University
Ottawa, Ontario K1S 5B6, Canada

sivarama@scs.carleton.ca

TECHNICAL REPORT TR-97-2

Abstract

Two typical architectures of parallel database systems are the shared-everything and shared-nothing architectures. Shared-everything architecture provides better performance than the shared-nothing architecture but it is not scalable to large system sizes. On the other hand, shared-nothing architecture provides good system scalability but is sensitive to data skew. Hierarchical architectures have been proposed to incorporate the best features of these two architectures (i.e., the shared-nothing and the shared-everything). In this paper, we present a detailed simulation study comparing the performance of the two-level hierarchical architecture with that of the shared-nothing and shared-everything architectures.

The results from the simulation experiments presented here show that a properly designed hierarchical system can provide performance very close to that of the shared-everything while providing system scalability similar to that provided by the shared-nothing architecture. The results also indicate that, when designing a hierarchical system, the cluster size should be maintained as large as the hardware limitations would allow. Then, the performance of the hierarchical system would be very similar to that of the shared-everything architecture under the system and workload models considered here.

Key words: Database systems, Parallel database systems, Performance evaluation, Simulation, Transaction processing.

1. Introduction

In recent years, the demand for high transaction processing rates has continued to increase substantially. At the same time, the amount of data involved in processing these transactions has also been increasing. One approach to meet this demand for increased transaction processing power is to parallelize transaction processing by utilizing multiple processors. In such parallel database systems, transactions are divided into several sub-transactions, all of which can be run in parallel.

A parallel database system can be built by connecting together several processors, memory units, and disk devices through a global interconnection network. Depending on how these components are interconnected, there can be a variety of possible architectures. Figure 1 illustrates two parallel database system architectural types: shared-nothing and shared-everything. Each of the architecture consists of processors (p), memory units (m), disk units (d), local buses, and a global interconnection network. In the shared-nothing (SN) architecture, each processor has its own (private) disk and memory units. A processor can communicate with other processors by means of explicit message passing. Shared-nothing database systems are attractive from the standpoint of scalability and reliability [Ston86]. However, load balancing is difficult to achieve, which results in high sensitivity of performance to data skew [Laks88]. Several commercial and experimental systems such as the Non-Stop SQL from Tandem [Tand88], DBC/1012 from Teradata [Tera85], Gamma at the University of Wisconsin [DeWi90], and Bubba at MCC [Jenq89] belong to this type of architecture.

In the shared-everything (SE) architecture, both memory and disk units are shared by all processors. As a result, this architecture allows easier load balancing, which in turn reduces the performance sensitivity to data skew. Thus, from the performance point of view, this architecture is desirable. However, these systems are less scalable and reliable compared to the shared-nothing systems because of the demands on communication, memory, and disk bandwidths.

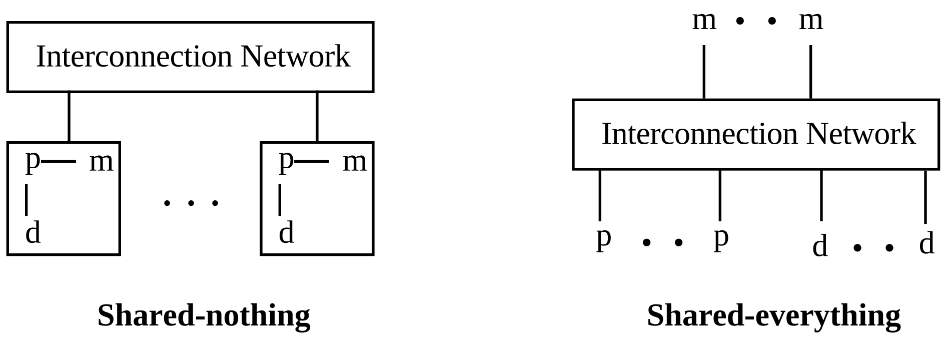


Figure 1 Shared-nothing and shared-everything parallel database system architectures

The suitability of various architectures for transaction processing has been studied by several researchers. Bhide and Stonebraker [Bhid88a] compared the performance of these architectures under a synthetic data management workload. Their results show that the shared-everything outperforms the shared-nothing by a fairly wide margin. However, performance is only one part of the equation. Other considerations, primarily scalability, hardware cost and reliability will tend to favour the shared-nothing approach. Although shared-nothing systems have the best scalability and reliability [Ston86] they are very sensitive to the data skew problem [Laks88]. Shared-everything allows the collaborating processors to share the workload more effectively; however, shared-everything suffers from the limitation of the communication, memory and disk I/O bandwidths. Similar concerns exist in the design of parallel systems (shared-memory versus distributed-memory systems). Hierarchical architectures have been proposed to incorporate the best features of the two architectures while minimizing the associated disadvantages. Examples include the Cedar system [Gajs83] and the Stanford Dash system [Leno92]. Hierarchical structures have also been proposed for parallel database systems[1] [Kits90, Hua91a[2]].

Hua et al. [Hua91a] present a hybrid structure in which shared-everything clusters are interconnected through a communication network to form a shared-nothing structure at the inter-cluster level. The key idea is to use shared-everything clusters to minimize the data skew effect and to use shared-nothing inter-cluster structure for good scalability. Figure 2 shows this architecture. Hua et al. discuss the optimum cluster size

[1]Actually, inspired by the Cedar multiprocessor system architecture, we have arrived at this type of hybrid architecture independently.

[2][Bhid88a, Bhid88b, Pira90] also suggest the use of a hybrid architecture as a compromise.

design issue by performing sensitivity analysis of a single-join operation to data skew and commuincation, memory, and disk bandwidths using a simple analytical performance model.

The Super Database Computer (SDC), which is a high-performance relational database server for a join-intensive environment, is also a hierarchical architecture of shared-nothing and shared-everything [Kits90]. It consists of several processing modules connected to each other through a message-passing interconnection network. Each processing module itself is designed as a tightly coupled multiprocessor system with good computational power. Some encouraging simulation results for single-join workload have been obtained for the SDC indicating that the hybrid architecture is a promising architecture.

Both of the studies mentioned above present the performance of a single-join for the two-level hierarchical architecture. Bhide [Bhid88b] provides the performance of shared-nothing and shared-everything under the assumption of all update-based transactions. Our work involves designing and implementing a multiple-architecture simulator, which provides a uniform test bed for shared-nothing, shared-everything and two-level hierarchical architectures. Using the simulator, we could explore the performance of shared-nothing, shared-everything and two-level hierarchical architectures under a variety of workloads such as multiple updates, multiple joins or mixed transactions, most of which have not yet been reported in the literature.

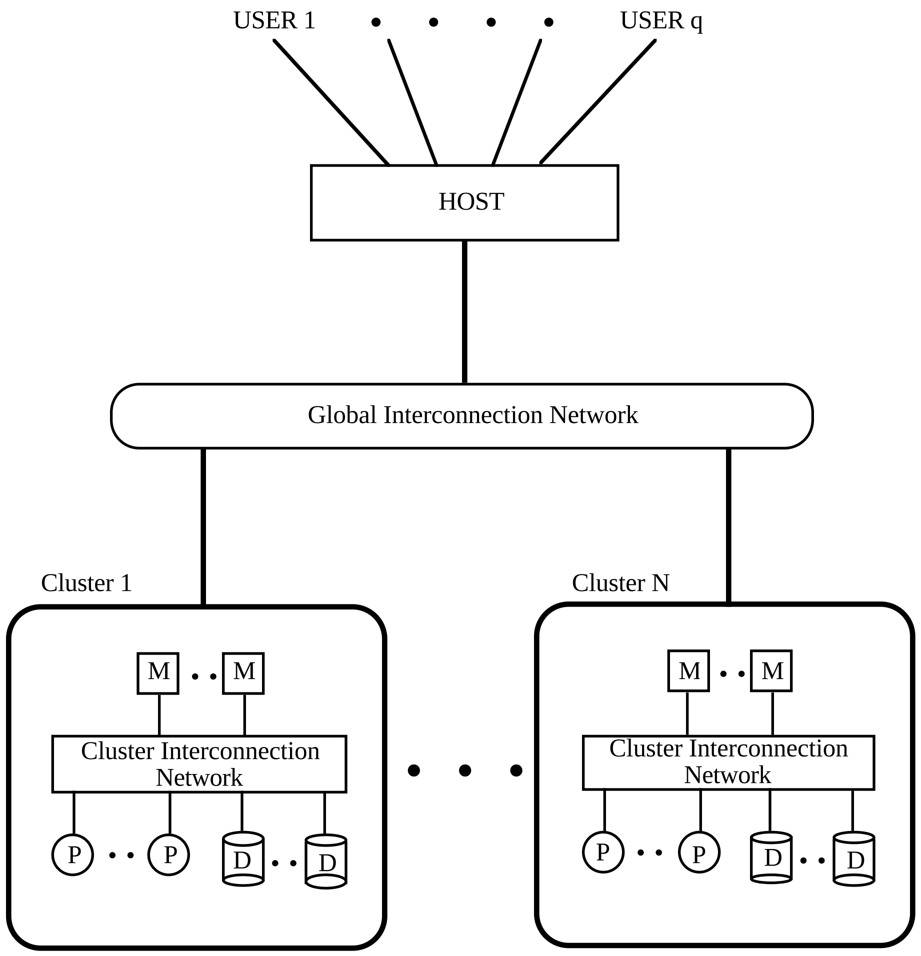

USER 1
USER q
HOST
Global Interconnection Network
Cluster 1
Cluster N
M
M
Cluster Interconnection Network
P
P
D
D

Figure 2 A two-level hierarchical architecture consisting of *N* shared-everything clusters that are inter-connected through the global interconnection network to form a shared-nothing structure at inter-cluster level.

In this paper, we study the performance of two-level hierarchical architectures and compare their performance to that of the shared-nothing and shared-everything systems under a set of reasonable assumptions. The remainder of the paper is organized as follows. Section 2 presents details about the simulation model, which includes the database, transaction, system, and queueing models. Section 3 discusses the parameter settings and provides a performance comparison of the shared-nothing, shared-everything, and the two-level hierarchical architectures. Section 4 concludes the paper by summarizing the results.

2. THE SIMULATION MODEL

Figure 2 shows the two-level hierarchical architecture system model. When a transaction is received form a user terminal, the host distributes the transaction to all the relevant clusters via the global interconnection network. Since the data is distributed across the clusters (actually across the disks) a transaction can be executed by all the clusters simultaneously. Each cluster maintains a queue of transactions and schedules a transaction for execution independent of the other clusters in order to maximize its access operations and to minimize its idle time. In addition, transactions are also also executed within the clusters concurrently. Host is mainly responsible for interfacing and preprocessing. The clusters perform the database operations.

2.1. The Database Model

We model the database as a collection of relations, each of which is declustered uniformly among all the disks. A set of tuples comprises a relation. The partition on each disk is defined by an attribute frequency distribution function and the distinct attribute size. We assume that the attribute frequency distribution function is either uniform or normal. We also assume that the distribution for every disk is the same and every distinct attribute appears in every disk. Therefore, from the attribute distribution and the distinct attribute size, we can easily derive the size of each attribute provided we correspond every real attribute to an abstract attribute: the ordinal number. For the uniform distribution, every attribute gets the same number of tuples; for the normal distribution, we assume that the middle attribute has the highest probability of appearing in the relation and the variance parameter of the normal distribution is then used to derive the probability of other attributes. If the attribute A with the smallest frequency p_A has x number of tuples then any attribute B with frequency p_B will have $ceil(p_B/p_A) * x$ number of tuples. The relevant parameters for the database model are as follows.

Num Relations: Number of relations in the database

AttributeSize$_i$: Number of distinct attributes in relation i

AttrFreqDist$_i$: The attribute frequency distribution function on each disk for relation i

RelationSize$_i$: Number of tuples in relation i (Note that this parameter determines the range of any relation, but the actual relation size is determined by its *AttributeSize* and *AttriSizeDist*.)

TupleSize: Bytes in a tuple (bytes/tuple)

PageSize: Number of tuples per page (tuples/page)

2.2. The Transaction Model

Transaction workload models are very important in performance evaluation. Most existing workload models for database system performance evaluation are too specialized to be adapted. Several researchers have modelled transactions as a sequence of read/write operations from the concurrency point of view [Agra87, Bhid87, Bhid88a, Bhid88b, Care84, Care88, Care89]. These models ignore the communication overhead among the processing modules, which is very important in parallel database systems. Hua et al. [Hua91a] used a single-join workload model to study the performance of the hybrid architecture. The emphasis of this study by Hua et al. has been on the communication overhead problem. Furthermore, they considered an analytical model for only a single-join operation and avoided the concurrency behaviour of the system. For example, what is the effect of several join transactions executing simultaneously in the system? What is the effect of mixed join and update transactions? Therefore, in our transaction workload model, we would like to introduce both the update-based transactions such as read/write sequence and query-based transactions such as join, selection, projection.

2.2.1. Update-based transaction model

The update-based transaction is just a sequence of read and write operations on pages because page is the most suitable unit for disk and CPU processing. The abstract form of each operation is <*Read* or *Write*, *RelationName*, *PageName*> whose meaning is to *read* or *write* a page *PageName* for *RelationName*. For the write operation, we assume any modified page has to be read before it is written, i.e., no blind write is allowed.

2.2.2. Query-based transaction model

Query-based operations are just read-only operations because the intermediate results are assumed to be kept in the temporary area of the disks and no permanent modification of the physical database is involved. In a parallel database system, database is usually declustered among multiple disks due to disk capacity limitation and for effective parallel processing. Partitioning techniques are particularly suitable for selection and projection operations, which can be processed locally. So they can be modeled by read-only update-based transactions. However, serious problems arise with the join operation, which incurs considerable communication overhead. Therefore, in the hierarchical and shared-nothing architectures, consideration of the join operation is necessary and important. For simplicity and without loss of generality, we only consider join transactions that perform a join operation between two relations.

For the join operation, we use a hash-based join as the join method in our two-level hierarchical architecture. The rationale for this is as follows. There are several relational join algorithms that are suitable for single processor as well as multiprocessor environments. Nested-loop, sort/merge and hash-based algorithms are the most commonly used algorithms. The analysis in [Laks88] shows that the nested-loop algorithm is useful only when the relations to be joined are relatively small. Furthermore, when the relations are large, hash-based algorithms are superior to sort/merge provided the relations are not already in sorted order or the final result need not be sorted before presenting to the user. Because parallel database systems are mainly aimed at very large relations hash-based join methods should work better.

Furthermore, Schneider and DeWitt [Schn89] have shown that, besides being faster under most conditions, hash-based join algorithms have the property of being easy to parallelize because tuples of a relation in one bucket are never joined with tuples of the other relation in another bucket and thus the corresponding buckets can be handled independently. After comparing the performance of four parallel hash-based join algorithms (Sort/merge with Hash, Simple Hash, Grace Hash-join and Hybrid Hash-join) in a shared-nothing environment, they showed that in the case where the join attribute values are highly skewed and the available memory relative to the size of the smaller relation is limited, the optimizer should choose a non-hash-based algorithm such as sort-merge. Actually, it is the shared-nothing architecture that is very sensitive to the data skew problem. If we could eliminate some of the negative effects of non-uniformly distributed inner relations with some other useful architecture such as the shared-everything (data skew has the least effect on the shared-everything) hash-based join method can still work well.

More importantly, there have been several hash-based join algorithms to deal with the data-skew problem such as Tuple Interleaving Parallel Hash Join (*TIJ*), Adaptive Load Balancing Parallel Hash Join (*ABJ*) and Extended Adaptive Load Balancing Parallel Hash Join (ABJ^+) [Hua91b]. The analysis in [Hua91b] has shown that these schemes are able to provide savings over the conventional parallel hash-based join algorithms even at low data skew conditions. In addition, Hua and Lee suggest that the *ABJ* algorithm should be used if the degree of skew is mild ($\sigma < 60\%$), where σ is called the degree of skew. The degree of data skew σ is defined as follows. For each of the relations participating in a join operation, the skewed processing node has $\sigma \times 100\%$ more tuples than each of the remaining processing nodes. The ABJ^+ algorithm should be used if the degree of skew is significant ($\sigma \geq 60\%$) and the *TIJ* algorithm should be used in lieu of ABJ^+ algorithm if the relations are very seriously skewed initially, and the communication bandwidth is sufficiently large. *TIJ* and ABJ^+ are better only in the high skew situations because they have the I/O and communication overheads associated with the deferred bucket allocation strategies, which are more serious than the mild data skew problem. If we could eliminate the partial I/O and communication overhead problems, *TIJ* and ABJ^+ can do better even if the degree of skew is mild.

Based on these observations, the hash-based join has been selected as the join algorithm in the two-level hierarchical architecture because the clusters of two-level hierarchical architecture are organized as shared-everything systems and ABJ^+ has been chosen for the two-level hierarchical architecture because the I/O and communication overhead among a cluster can be reduced.

2.3. System Model Assumptions

The following assumptions are made about the system. Each data item is stored at exactly one site (i.e., no replication of data). Relations are uniformly distributed among all the disks in the system. Each transaction consists of multiple processes that can be executed at multiple sites and the actual workload depends on the data selected. Sufficient buffer space exists to allow the retention of updates until commit time and a log-based recovery scheme is assumed so that only the log pages must be forced prior to commit.

Two-phase locking is assumed to be the concurrency control method. Dynamic locks, in which the locks are requested just before accessing the data, are implemented by the simulator. Once the locks are acquired by a transaction the locks are held until the end of the transaction. Page level locking granularity is assumed in the experiments. We consider only read and write locks, with the proviso that a read lock can later be upgraded to a write lock. Note that we do not allow blind writes i.e., a page must be read before attempting to write into it.

Two-phase commit protocol is assumed. There are a number of communication paradigms that can be employed to implement a two-phase commit protocol. These are the centralized two-phase commit protocol, linear two-phase commit protocol, and the distributed two-phase commit protocol. We use the distributed two-phase commit protocol as the two phase commit protocol in the two-level hierarchical architecture because it eliminates the need for the second phase of the protocol since the participants can reach a decision on their own. Certainly, the distributed two-phase commit protocol incurs more message passing overhead than the centralized two-phase commit protocol. Considering that the join operation also employs an intensive data distribution process, it can be expected that the workload for the communication network can be much more uniform if both the intensive processes could be done intermittently.

2.4. The Queueing Model

The queueing model for the two-level hierarchical database system is shown in Figure 3. When a transaction is initiated by a user terminal, the transaction type, the relations and the pages accessed are chosen by the transaction generator. The transaction generator generates the workload for the host. There are two types of transactions: update-based and query-based transactions. For each simulation run, the distribution for both types is determined. The selection proportion is used to determine whether the transaction is update-intensive or query-intensive. We assume that the relations involved in both types of transactions are uniformly distributed among the total number of relations.

The transaction generated by the transaction generator waits in the global transaction ready queue until the number of active transactions in the system falls below the degree of multiprogramming (*mpl*). The value of *mpl* ranges from one to the number of processing modules. Global transaction manager (GTM) is the focus of the host queueing model.

The host splits the user transaction into a set of cluster transactions and distributes the cluster transactions to all the relevant clusters via the global interconnection network since the data is distributed across the clusters (actually across the disks). All the cluster transactions can be executed simultaneously. Each cluster maintains a queue of the cluster transactions and splits every cluster transaction into processor-oriented sub-transactions again. Each sub-transaction is scheduled for execution concurrently within the clusters.

2.4.1. Global transaction manager (GTM)

Once a transaction enters the system, GTM loads a global coordinator for the transaction at the host node. For the update transaction, GTM will split the transaction into cluster-related sub-transactions, which are transferred to the appropriate CTMs (Cluster Transaction Manager) to initiate the global cohorts. For the join transaction, GTM broadcasts the transaction to all cluster CTMs to initiate its global cohorts. The GTM will then assign one global cohort as the global coordinator's subhost for coordination and will keep all the related information such as the transaction id, subhost id, the sub-transaction workload and the participants' list etc. in the global coordinator and the log. Finally, GTM will send all the sub-transactions, each of which includes the necessary information such as transaction id, subhost's id and participants' list etc. to the appropriate clusters.

When a global coordinator gets the message "Transaction-Done" from a CTM it writes a commit record in the log and informs the completion of the transaction to the user terminal that initiated the transaction. The terminal will initiate another new transaction after "think time."

When a global coordinator receives the message "Transaction-Abort" from a CTM, it writes an abort record in the log, delays for a period of time Delay-time and then moves the transaction into the Global Transaction Ready Queue with high priority for re-execution. However, the transaction's workload is preserved.

2.4.2. Global network manager (GNM)

The model of the global interconnection network is encapsulated by the GNM. In our network model, we take network communication bandwidth as the only parameter for measuring messages passing among the host and the clusters. The message-passing time within the network is calculated using the network bandwidth and the message size.

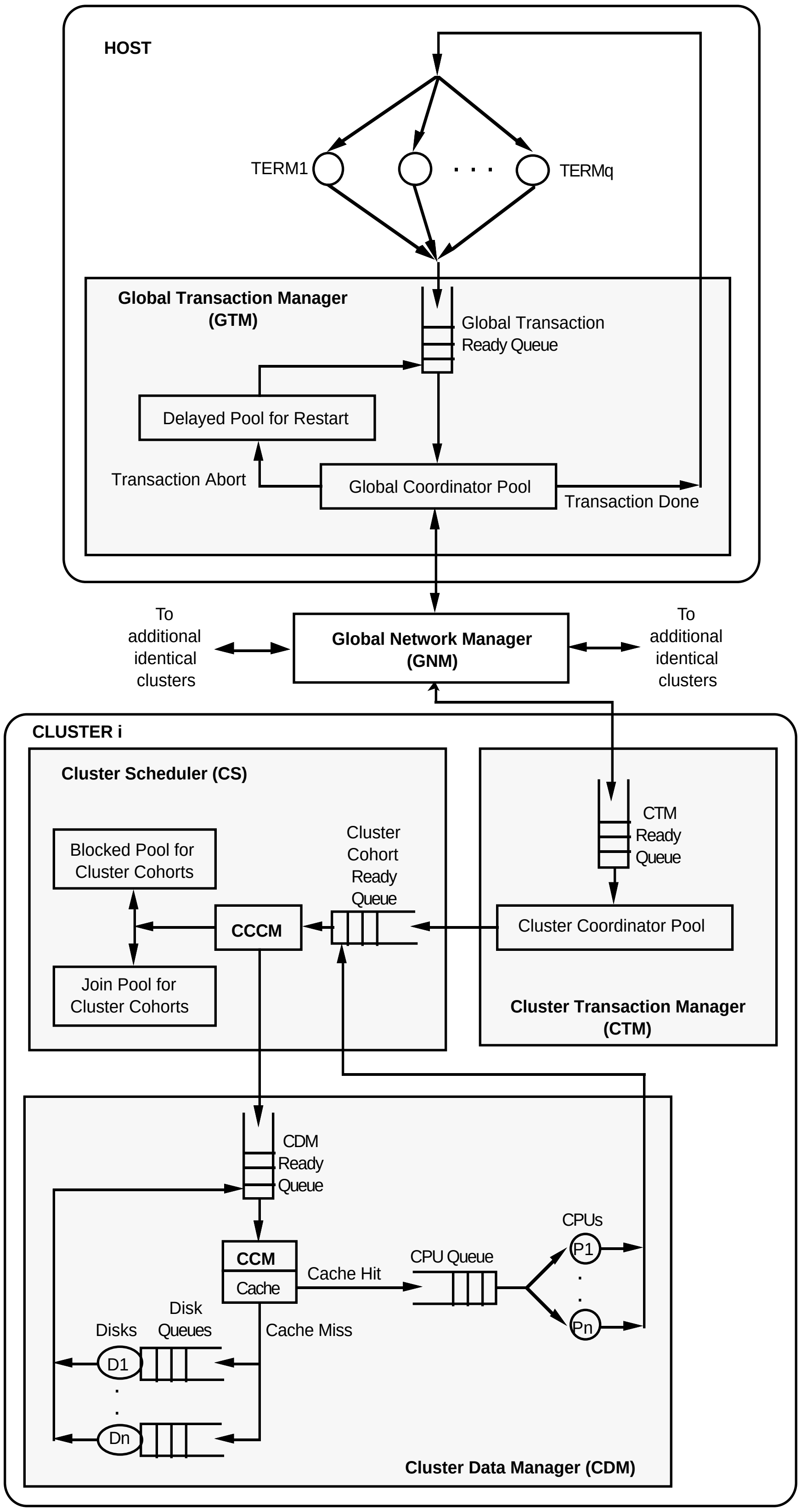

HOST
TERM1
. . .
TERMq
Global Transaction Manager (GTM)
Global Transaction Ready Queue
Delayed Pool for Restart
Transaction Abort
Global Coordinator Pool
Transaction Done
To additional identical clusters
Global Network Manager (GNM)
To additional identical clusters
CLUSTER i
Cluster Scheduler (CS)
Blocked Pool for Cluster Cohorts
Join Pool for Cluster Cohorts
CCCM
Cluster Cohort Ready Queue
CTM Ready Queue
Cluster Coordinator Pool
Cluster Transaction Manager (CTM)
CDM Ready Queue
CCM
Cache
Cache Hit
Cache Miss
CPU Queue
CPUs
P1
Pn
Disks
Disk Queues
D1
Dn
Cluster Data Manager (CDM)

Figure 3 Queuing model for the two-level hierarchical architecture

2.4.3. Cluster transaction manager (CTM)

When a CTM receives a cluster transaction from the CTM Ready Queue it initiates a global cohort, which is actually a local cluster coordinator for the cluster transaction. The cluster transaction is split into a number of sub-transactions depending on the transaction type and the data items accessed. Each sub-transaction corresponds to a cluster cohort. For the update type of transaction, the transaction is split into the processor-related sub-transactions. (Note that we only consider one sub-transaction per disk.) For the join type of transaction, each disk has a sub-transaction which is responsible for the split phase in ABJ^+ (Note that, within a cluster, we assume that the matching tuples have been distributed to the corresponding disks after the split phase).

The centralized two-phase commit protocol is adopted in the system. The selection of the center is based on the round-robin strategy so that the workload could be balanced among the clusters. Because of the relatively low probability of deadlock [Gray81] and the substantial communication overhead involved for large systems, global periodic deadlock detection scheme is used. When a deadlock occurs, a transaction that has obtained minimum locks (called the min-lock strategy) will be selected as the victim[3]. The reason for the selection of the min-lock strategy is as follows. Usually, a larger transaction tends to request more locks than a small transaction. Aborting a transaction with the minimum locks implies aborting a transaction that has completed the minimum amount of work. The period of the global deadlock detection varies between an upper and a lower limit as in the Gamma [DeWi90]. Within these bounds, the global deadlock detection invocation period is doubled (halved) if there was no deadlock (deadlock) the last time it was invoked. In addition, to balance the workload generated by the global deadlock detection, every cluster takes turn as the deadlock detection coordinator.

2.4.4. Cluster scheduler (CS)

Cluster scheduler handles concurrency control requests from cluster cohorts, requests to force some cluster cohorts to abort and requests for cluster cohorts to commit etc. CS also informs the CTM with messages such as “Work-Done”, “Split-Done”, “Done” and so on.

Cluster cohorts require read locks on items that they read, converting their read locks to write locks on items that need to be updated. (Note that, in an update type of transaction, all reads happen before all writes.) As usual, read locks can be shared but write locks are exclusive. Locks are set dynamically, as the cluster cohort executes, with the cohort blocking until the next item to be accessed has been successfully locked. Locks are held until the transaction has been successfully committed or aborted. The blocked queue holds the cluster cohorts that are waiting for locks. Of course, local deadlocks are possible. Local cluster deadlock avoidance strategy is applied whenever two cluster cohorts want to write on the shared lock page. The cohort with the later write-request will be aborted.

2.4.5. Cluster data manager (CDM)

Cluster data manager receives the operation of granted cluster cohort from the CS and processes it by manipulating storage. Any granted cluster cohort turns to the cluster cache manager (CCM) which will direct the cluster cohorts to disk server for cache non-hit and to CPU servers for cache hit. We will adopt a no-redo/no-undo recovery strategy for update-based transactions in our model, which is based on [Elha84]. There is a single queue for all CPUs (because of the shared-everything clusters) since the scheduler assigns a job to the next available CPU. There is a single queue for each disk because the data is disk-oriented.

[3] Note that more than one transaction abort may be needed to break a deadlock.

3. SIMULATION EXPERIMENTS AND ANALYSIS

3.1. Parameter Settings

We assume, by default, that there are 128 terminals attached to the host and there are 8 processing modules in the system. This small number is used mainly because of the long simulation runs to generate the confidence intervals. Every processing module has a processor, a memory unit and a disk. We can cluster processing modules to obtain different architectures. For example, clustering two processing modules gets four clusters, each of which has two processing modules. Table 1 lists different possible architectures with 8 processing modules. We use the notation 8-8-1, 8-4-2, 8-2-4 and 8-1-8 to represent these architectures. The 8-8-1 system represents the shared-nothing (SN) architecture. The architectures 8-4-2 and 8-2-4 belong to the two-level hierarchical architectures (2HAs). The shared-everything (SE) is represented by 8-1-8. We will use the simulation results to compare the performance of the SN, 2HAs and the SE architectures. In order to compare the performance of these architectures, the parameter settings should be consistent. Table 2 gives the default simulation parameter values used in the experiments.

Table 1 The different architectures considered for a system with 8 processing modules

	SN	2HA1	2HA2	SE
Number of clusters	8	4	2	1
Cluster size	1	2	4	8

3.1.1. Database and transaction parameters

We assume that there are 32 relations, each of which has 100 pages on each disk. So the total database size in the system is 32*8*100 = 25,600 pages. It is also assumed that every relation has the same number of attributes and every attribute has the same number of tuples except when static data skew effects are considered. Static data skew can be defined with the attribute normal distribution. The degree of static data skew depends on the variance of the normal distribution and the attribute size. In our simulation, we assume that the skewed attributes have more tuples than other, non-skewed, attributes and the number of tuples for the skewed and non-skewed attributes are uniformly distributed among all the disks.

The relation selection is uniform among all the 32 relations. The number of pages read by an update-based transaction is determined by the number of read operations for each update-based transaction. Each page belongs to a selected relation. The write probability decides the number of pages that will be updated by a transaction. For every join-based transaction, two relations are selected.

For join-based transactions, the dynamic data skew after hashing is important. We will adopt a definition of data skew degree that is similar to that in [Hua91b] for the SN architecture to investigate the impact of data skew on the performance of SN, SE and 2HA architectures.

In the case of SN, the following definition of data skew is used.

T :	Relation size in pages for the two join relations;
Ps :	Size in pages of the skewed partition;
Pu :	Size in pages of each of the remaining unskewed partitions;
N :	Number of processing modules;
σ :	The degree of data skew which is defined as

Table 2 Default parameter settings used in the simulation experiments

Number of terminals (TN)	128
Number of processors (processing module, PM)	8
Number of clusters (CN)	8, 4, 2, 1
Number of processors within a cluster (C)	1, 2, 4, 8
Tuple size	200 bytes
Page size	1000 bytes
Number of pages for each relation on each disk	100 pages
Number of relations	32
Access probability for each relation	1/32
Number of read operations for each update-based transaction (TS)	32
Write probability for update-based transactions	1/4
Processing rates of GTM, CTM, CS and Cache capacity	infinity
CPU processing rate	1 MIPS
Network bandwidth	1 Mbytes/sec
Time for processing a R/W page	22.5 ms
Time for reading a join bucket	2.8 ms
Time for reading and writing a join page	5.6 ms
Instructions for processing a R/W page	2K
Instructions for processing a join page	4K
Instructions for processing a new join bucket	4K
Cache capacity of a cluster	TS/CN*TN
Multiprogramming level (*mpl*)	1 ... PM
Mean of think time distribution, Uniform	0 ... 100 sec
Mean of delayed time distribution, Uniform	1 sec
Upper bound of deadlock detection	60 sec
Lower bound of deadlock detection	5 sec
Bytes required to send a simple message such as "Vote-Commit" etc.	32 bytes
Hashed database size information for join	400 bytes

$$\sigma = \frac{|Ps - Pu|}{Ps}$$

We also assume that there is only one single skewed processing module. The remaining nodes are unskewed and have the same number of data pages. This assumption is based on the fact that the most seriously skewed processing module constitutes the bottleneck in the SN architecture. Then, the following should hold.

$$Ps + (N-1) \times Pu = T \Rightarrow \begin{cases} Ps = \dfrac{T}{1+(N-1)\ (1-\sigma)} \\ Pu = \dfrac{(1-\sigma)\ T}{1+(N-1)\ (1-\sigma)} \end{cases}$$

Similarly in the case of 2HAs, *Ps* needs be redefined as follows because of the SE cluster. In SE, only the skewed pages are amortized among all the processing modules within a cluster.

Ps : Size in pages of the skewed partition for each processing module within a skewed cluster.

Therefore,

$$Ps = \frac{|T - Pu \times (N - C)|}{C}$$

where *C* is the number of processing modules within a cluster.

In our simulator, the bucket size is equal to the page size. So *Ps* and *Pu* define the number of join buckets for every processing module.

3.1.2. System parameters

The processing rates for GTM, CTM, CS and Cache, which are common to all the architectures, can be infinitive without destroying the consistent behavior among the different architectures. GTM is same for any architecture. Although different architectures have different number of CTMs, CSs and Caches, the overall rates should be consistent. For example, 8-8-1 has eight CTMs and 8-4-2 has four CTMs. But the CTM for 8-4-2 should be twice faster than that for 8-8-1. Therefore, we assume all the rates for GTM, CTM, CS and Cache are infinitive for simplicity.

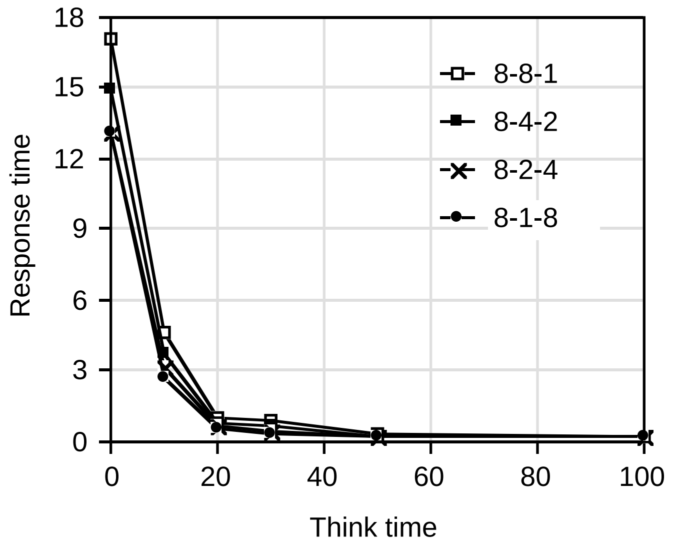
We also assume that the disk time to read or write a page for the update-based transactions is 22.5 ms, the disk time for reading a join bucket is 2.8 ms and the disk time for reading and writing a join page is 2*2.8=5.6 ms. The calculation of disk time is based on disk's seek time, rotational delay and block transfer time. For each page access of update-based transactions, all three factors should be considered because of the randomness. But, for the join-based transactions, the seek time and rotational delay can be minimized if the pages or buckets are consecutively stored on the permanent or temporary disk area and the bulk of data pages are prefetched into the cache awaiting processing.

3.2. Performance with All Update-Based Transactions

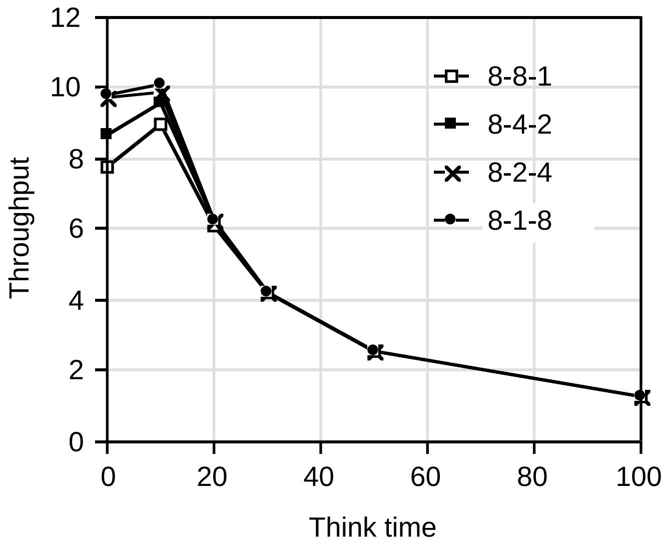
Performance of the four architectures with all update-type transactions is shown in Figure 4. The multiprogramming level *mpl* is fixed at 4 for all these results.

From Figure 4a it can be seen that the shared-nothing (i.e., 8-8-1 system) provides the worst performance while the shared-everything (i.e., 8-1-8 system) provides the best performance. This is an expected result.

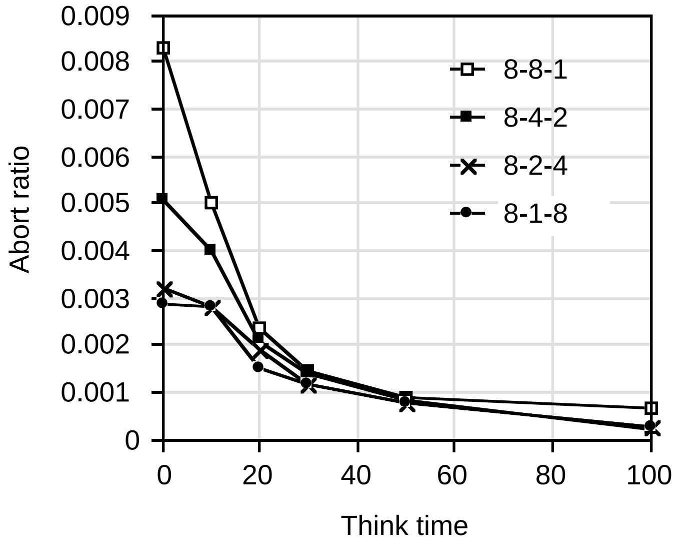
The performance of the hierarchical architectures is intermediate between that of the shared-nothing and shared-everything. In particular, the performance of the 8-2-4 HA is very close to that of 8-1-8 SE architecture. These differences in performance are evident only in the lower think time range, where the system load is sufficiently high to generate a significant amount of data contention. These results show that the hierarchical architectures improve the performance over the shared-nothing architecture.



(a) Response time versus think time



(b) Throughput versus think time



(c) Abort ratio versus think time

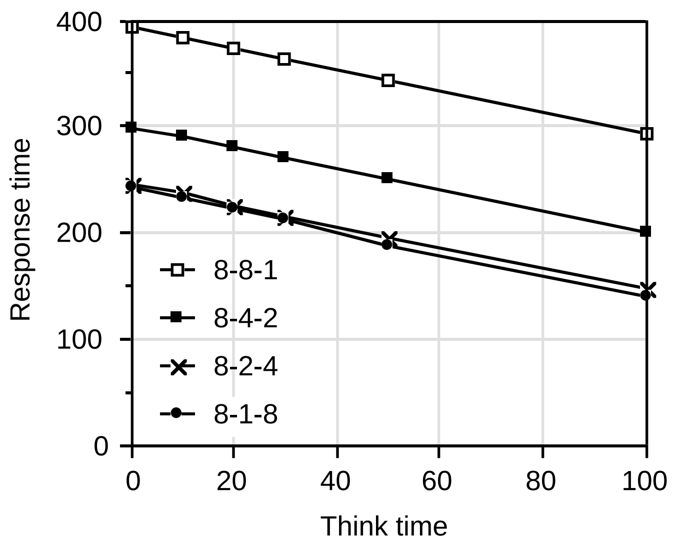
Figure 4 Performance of all-update transactions

Figure 4b shows the throughputs associated with the four architectures. It can be seen from this figure that all the throughputs increase initially as the think time is increased from 0 to 10. This is due to the decrease in abort ratio as shown in Figure 4c. Further increases in think time decreases the throughput as the system load decreases. The initial increase in throughput of the 8-8-1 SN system is higher than that associated with the other systems. This is because the 8-8-1 SN system induces higher transaction aborts as shown in Figure 4c.

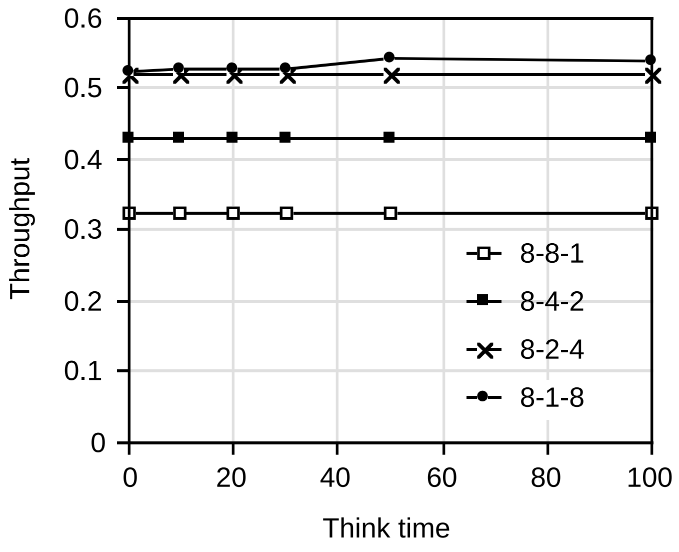
From the data presented here, it may be noted that, while the SE architecture is the preferred choice, a properly designed HA can also provide a performance comparable to that of the SE architecture. As one might expect, clustering more processing modules gives a better performance. The cluster size is a function of various system parameters such as the communication bandwidth, memory bandwidth, etc. The analysis provided in [Hua91a] can be used to arrive at the high level design issues such as the optimum cluster size etc. The emphasis of this paper is to provide a performance comparison of the four architectures, assuming that there are no bottlenecks in the system.

3.3. Performance with All Join-Based Transactions

The response time and throughput for all join-based transactions are presented in Figure 5. These results are for $\sigma = 0.9$ and *mpl* = 4. The flat throughput (Figure 5b) indicates that the system is in a stable state as long as there are enough transactions waiting for processing regardless of the increase of think time. The response time in Figure 5a decreases linearly by the amount of think time as the think time increases. This is simply because the transactions would have to wait to get to the processing stage. However, if we continue to increase the think time of the transactions until such point that there are not enough transactions waiting to be processed, the response time and the throughput will decrease significantly.



(a) Response time versus think time



(b) Throughput versus think time

Figure 5 Performance of all-join transactions

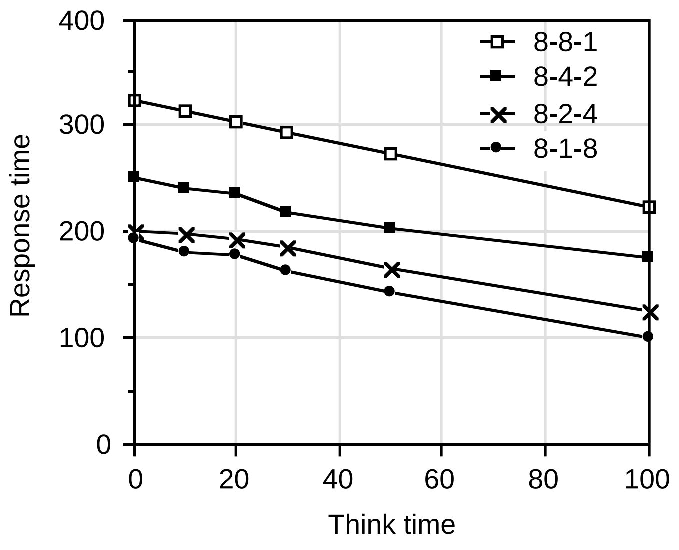
The two hierarchical systems provide substantial improvements in performance of join operations compared to the shared-nothing system. Also note that the performance of the 8-2-4 HA system is similar to that of the 8-1-8 SE system. This improvement in performance is due to load balancing and reduced communication overhead. This further suggests that HAs can be designed to provide performance very close to that provided by the SE architecture.

3.4. Performance Sensitivity

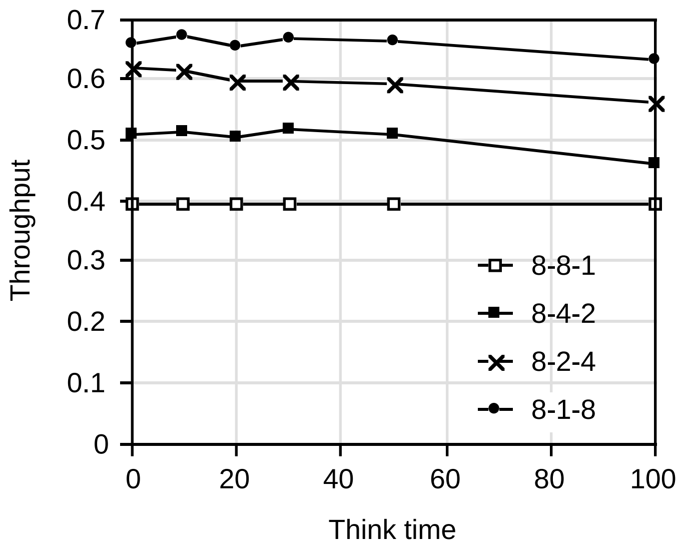
In this section, we assume a mixed transaction workload with 20% update-based transactions and 80% join-based transactions. This section investigates the impact of the mixed transactions, data skew, system size, and the database size on the performance of the four systems.

3.4.1. Sensitivity to mixed transactions

In the mixed transactions, there are 80% join transactions, which dominate the system execution time. Therefore, the performance of the four systems is similar to that for all join-based transactions (see Figure 6). Although there are 20% update-based transactions, which can cause transaction aborts, the abort ratio (not shown here) is less than that for all update-based transactions. Thus, for all the architectures considered here, the response time and throughput are better than those for all join-based transactions. Comparing Figures 5 and 6 confirms the above observation. Another observation from these figures is that the reduction in response time for the 8-8-1 SN system is more than that for the 8-2-4, 8-4-2 HA systems and the 8-1-8 SE system. This, as indicated before, is due to reduced communication overhead. If we change the proportion in the mixed transaction workload, we predict that the performance of any architecture will be between that for all-updates and that for all-joins of the corresponding architecture.



(a) Response time versus think time

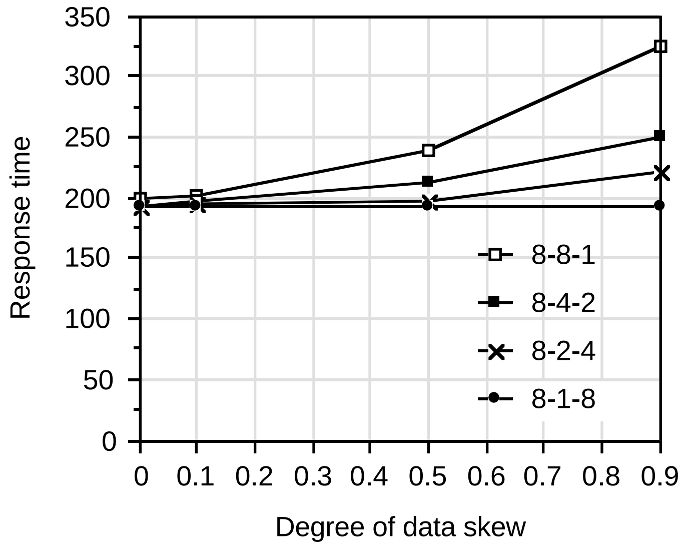


(b) Throughput versus think time

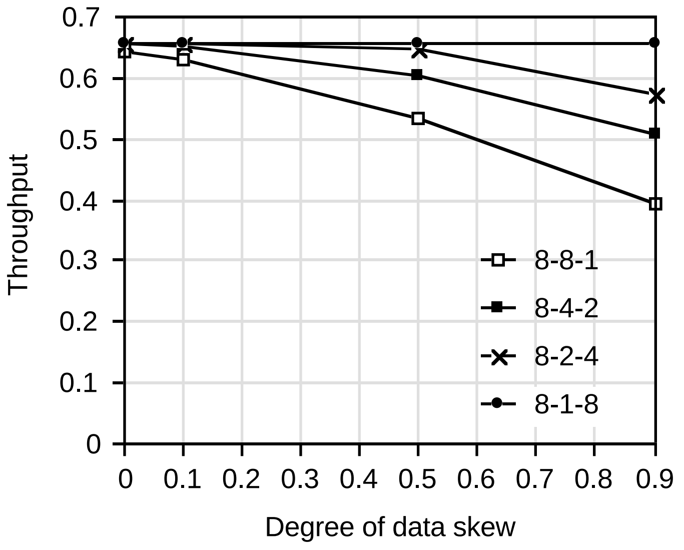
Figure 6 Performance of mixed transactions

3.4.2. Sensitivity to data skew

This section investigates the data skew impact on the performance of the four systems for the mixed transaction workload with *mpl* =4 and *thinktime* = 0. Figure 7a shows the response time results and the throughput is shown in Figure 7b. It can be seen from these figures that the 8-8-1 SN system is the most sensitive to data skew among the systems considered in this study. The response time at $\sigma = 0.9$ is 35% more than that at $\sigma = 0.5$, which in turn is 18% more than that at $\sigma = 0.1$ for the 8-8-1 SN system. The corresponding values for the 8-4-2 HA system are 19% and 7.79% respectively. These values reduce further for the 8-2-4 HA system to 13% and 1.29% respectively. Obviously, the performance of the 8-2-4 and 8-4-2 HA systems is closer to that of the 8-1-8 SE system than that for the 8-8-1 SN system, which indicates that the data skew has the least influence on 8-2-4, followed by 8-4-2, and followed by 8-8-1 among the non-SE systems considered in this study.



(a) Response time versus data skew



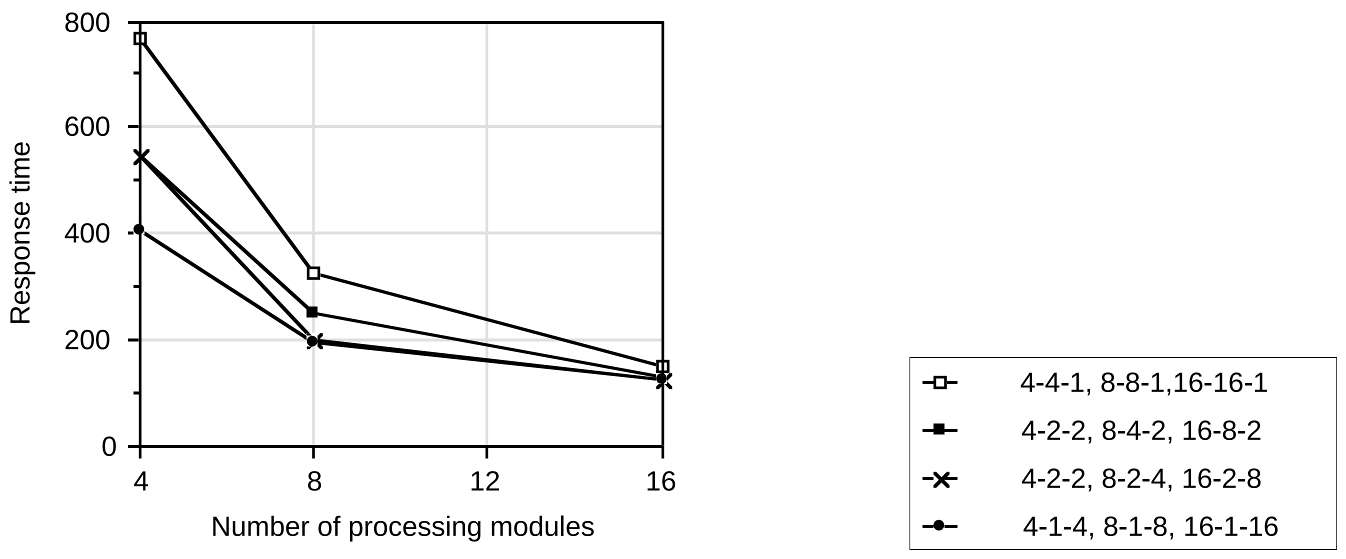
(b) Throughput versus data skew

Figure 7 Data skew effect on system response time and throughput with mixed transactions

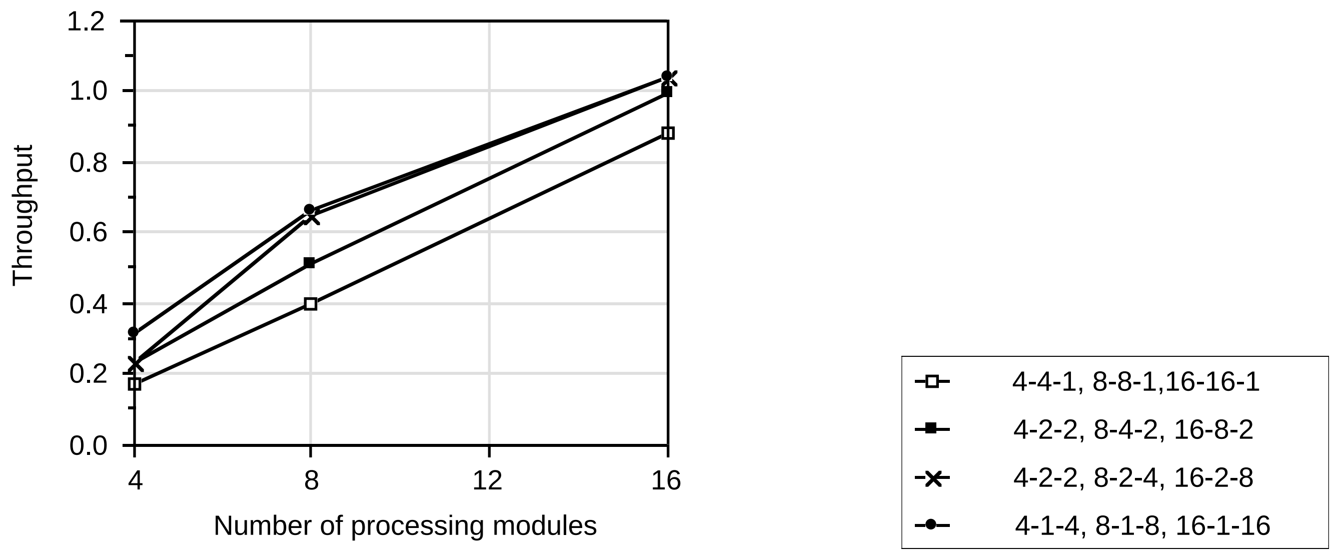
3.4.3. Sensitivity to system size

The impact of system size on the performance of different architectures is considered in this section. We have run several simulations for 4-4-1, 4-2-2 and 4-1-4 with *mpl* = 2, as well as 16-16-1, 16-8-2, 16-2-8 and 16-1-16 with *mpl* = 8 maintaining the same database size. The simulation results of the response time and throughput are shown in Figures 8a and 8b, respectively.

Because the database size is kept the same for all system sizes, maintaining the data skew σ= 90% will cause less skewed partitions for larger systems than those for smaller systems. Thus, the data skew effect will be smaller with increasing system size, which explains why the curves for the SN, 2HAs and the SE approach together with increasing number of processing modules.



(a) Response time versus system size



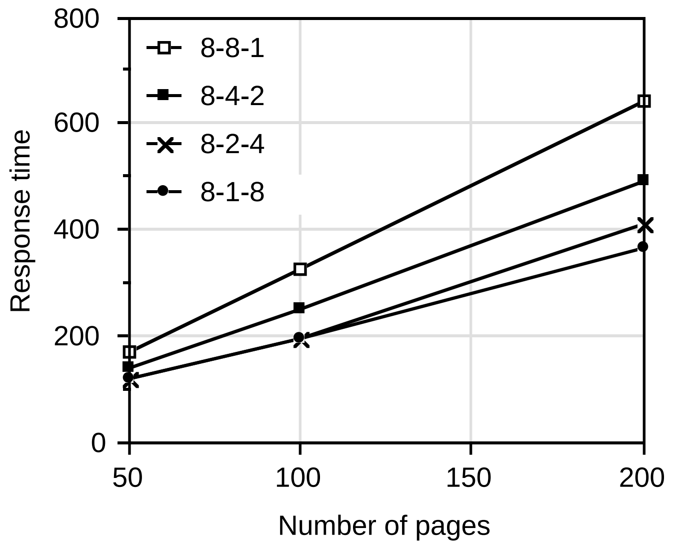
(b) Throughput versus system size

Figure 8 Impact of system size on system performance with mixed transactions

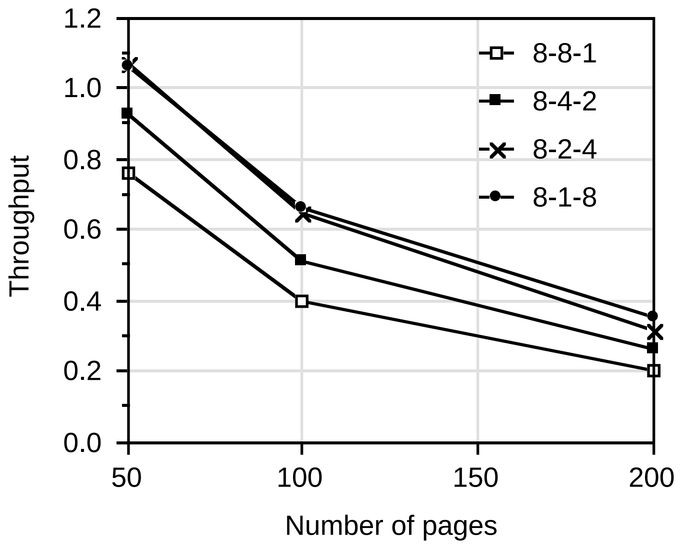
Comparing the performance difference at 4 processing modules and at 8 processing modules, it can be seen that the response time reduces by 58% and the throughput increases by 138% for 8-8-1 SN system; and the corresponding values for 8-1-8 SE system are 52% and 107%. Comparing the performance at 8 processing modules and at 16 processing modules we could see that, for 8-8-1 SN system, the response time reduces by 55% and the throughput increases by 123%; and the corresponding values for 8-1-8 SE system are 37% and 60%. The performance of 8-4-2 and 8-2-4 HA systems falls between that of 8-8-1 and 8-1-8. The data shows that the decreasing rate of response time for SN is more than those for 2HAs and SE, which indicates that SN is more sensitive to the system size and data skew than the 2HAs and the SE systems.

3.4.4. Sensitivity to database size

Due to limited simulation resources, the system size could not be large. Thus, in our default settings, the number of pages for each relation on each disk is set to 100. This section studies how the database size affects the performance of the different architectures. Figures 9a and 9b show the response time and throughput results, respectively. These results indicate that the rate of increase in the response time for 8-2-4 and 8-4-2 is always smaller than that for the 8-8-1 within the range of the database sizes considered. From the results, we predict that the difference in the performance of the 2HA and SN architectures increases with the database size.



(a) Response time versus database size



(b) Throughput versus database size

Figure 9 Impact of database size on system performance with mixed transactions

4. Conclusions

Shared-everything architecture provides a better performance than the shared-nothing architecture but it is not scalable to large system sizes. On the other hand, shared-nothing architecture provides good system scalability but is sensitive to data skew. Hierarchical architectures have been proposed to incorporate the best features of these two architectures (i.e., the shared-nothing and the shared-everything). We have presented a detailed simulation study comparing the performance of the two-level hierarchical architecture with that of the shared-nothing and shared-everything architectures.

The results from the simulation experiments presented here show that a properly designed hierarchical system can provide performance very close to that of the shared-everything while providing system scalability similar to that provided by the shared-nothing architecture. The results also indicate that, when designing a hierarchical system, the cluster size should be maintained as large as the hardware limitations would allow. Then, the performance of the hierarchical system would be very similar to that of the shared-everything architecture under the system and workload models considered here.

The simulation results presented here are obtained under certain assumptions about the workload and the system. Although these assumptions are reasonable, more simulation experiments are needed to test the impact of some of these assumptions. We have studied the performance influence of the system size and the database size. A natural question that arises is how the granule size, transaction size and transaction service time affect the performance of the different architectures? For a given architecture, with a fixed database size, granule size and transaction size, how will the multiprogramming level affect the performance and when will thrashing occur? For a dynamic workload, how will multiprogramming level adapt to achieve the best performance? In addition, we could explore the impact of different concurrency control methods and various hash-based join algorithms on the performance of these architectures. Future work will look at some of these issues in detail.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada and Carleton University. A shorter version of this paper appears in *Proc. Int. Conf. Computers and Their Applications*, Tempe, Arizona, March 1997.

REFERENCES

[Agra87] Agrawal, R., Carey, M.J., and Livny, M., “Concurrency Control Performance Modeling: Alternatives and Implications", *ACM TODS* ,12, 4 (December 1987), pp. 609-654.

[Bhid87] Bhide, A. and Stonebraker, M., “Performance Issues in High Performance Transaction Processing Architectures", *Proc. 2nd International Workshop on High Performance Transaction Systems*, Asilomar, September 1987. Also in *Lecture Notes in Computer Science*, Vol. 359, D. Gawlick, M. Haynie, A. Reuter (Eds.), Springer-Verlag, 1989, pp. 277-299.

[Bhid88a] Bhide, A. and Stonebraker, M., “A Performance Comparison of Two Architectures for Fast Transaction Processing", *IEEE Conference on Data Engineering,* 1988, pp. 536-545.

[Bhid88b] Bhide, A., “An Analysis of Three Transaction Processing Architectures", in *Proc. VLDB Conference*, Los Angeles, California, 1988, pp. 339-350.

[Care84] Carey, M., and Stonebraker, M., “The Performance of Concurrency Control Algorithms for Database Management Systems", *Proc. 10th VLDB Conference*, Singapore, August 1984, pp. 107-118.

[Care88] Carey, M.J., and Livny, M., “Distributed Concurrency Control Performance: A study of Algorithms, Distribution, and Replication", *Proc. VLDB Conf.*, Los Angeles, California, 1988, pp. 13-25.

[Care89] Carey, M.J., and Livny, M., “Parallelism and Concurrency Control Performance in Distributed Database Machines", *Proc. 1989 ACM SIGMOD Inter. Conf. on the Management of Data*, Portland, Oregon, Vol. 18, No. 2, June 1989, pp. 122-133.

[DeWi90] DeWitt, D., Ghandeharizadeh, S., Schneider, D., Bricker, A., Hsiao, H.-I, and Rasmussen, R., “The Gamma Database Machine Project", *IEEE Trans. on Knowledge and Data Engineering,* Vol. 2, No. 1, March 1990, pp. 44-61.

[Elha84] Elhardt,K., and Bayer,R., “A Database Cache for High Performance and Fast Restart in Database", *ACM Transactions on Database systems,* Vol. 9, No. 4, December 1984, pp. 503-525.

[Gajs83] Gajski, D., Kuck, D., and Sameh, A., “Cedar - A Large Scale Multiprocessor", *Proc. Int. Conf. Parallel Processing*, August 1983, pp. 524-529.

[Gray81] Gray, J., et al., ``A Straw Analysis of Probability of Waiting and Deadlock", *IBM Research*, RJ 3066, San Jose, Ca., Feb. 1981.

[Hua91a] Hua, K.A., Lee, C., and Peir, J.-K., “Interconnecting Shared-Everything Systems for Efficient Parallel Query Processing", *Proc. First Int. Conf. on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1991, pp. 262-270.

[Hua91b] Hua, K.A. and Lee, C., “Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning", in *Proceedings of the 17th International Conference on VLDB, Barcelona,* September 1991, pp. 525-535.

[Kits90] Kitsuregawa, M., and Ogawa, Y., “Bucket Spreading Parallel Hash: A New, Robust, Parallel Hash Join Method for Data Skew in the Super Database Computer (SDC)", in *Proceedings of the 16th VLDB Conference*, Brisbane, Australia 1990, pp. 210-221.

[Jenq89] B. C. Jenq, B. C. Twichell, and T. Keller, "Locking Performance in a Shared Nothing Parallel Database Machine," *IEEE Trans. Knowledge and Data Eng.,* Vol.1, No. 4, December 1989, pp. 530-543.

[Laks88] Lakshmi, M.S. and Yu, P.S., “Effect of Skew on Join Performance in Parallel Architecture", In *Proceedings of International Symposium on Databases in Parallel and Distributed Systems*, Austin, Texas, December 1988, pages 107-117.

[Leno92] Lenoski et al., “The Stanford Dash Multiprocessor,” *IEEE Computer*, March 1992, pp. 63-79.

[Pira90] Pirahesh, H., Mohan, C., Chang, J., Liu, T.S., and Selinger, P., “Parallelism in Relational Data Base Systems: Architectural Issues and Design Approaches", RJ7724(71589), IBM Almaden Research Center, September 1990.

[Schn89] Schneider, D., and DeWitt, D., “A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", *Proc. 1989 ACM SIGMOD Intern. Conf.*, 1989, pp. 110-121.

[Ston86] Stonebraker, M., “The Case for Shared Nothing", *Database Engineering*, 9, 1, March 1986, pp. 4-9.

[Tand88] The Tandom Performance Group, "A Benchmark of Non-Stop SQL on the Debit Credit Transaction," *Proc. ACM SIGMOD Conf.,* Chicago, Ill., June 1988, pp. 337-341.

[Tera85] Teradata DBC/1012 Data Base Computer Systems Manual, Release 1.3, Teradata Corp., Document No. C10-0001-00, February 1985.