http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 7 (7): 983-991, 2008
ISSN 1812-5638

A New Classification for Architecture of Parallel Databases

Pushpa Rani Suri and Sudesh Rani
Department of Computer Science and Applications, Kurukshetra University,
Kurukshetra-136119, Haryana, India

Abstract: Exploiting parallelism is the key to building high performance database systems. Several approaches to building database systems that support both inter and intra-query parallelism have been proposed. These approaches can be classified as either Shared Nothing (SN) or Shared Everything (SE). Although the SN approach is highly scalable, it requires complete data partitioning and tuning to achieve good performance whereas the SE approach suffers from non scalability. We propose a scalable sharing approach which combines the advantages of both SN and SE. Requirements to parallel database systems are formulated, which serve as criteria for comparing various architectures. We analyze the performance of our approach and compare with that of a SN and SE system. We find that our approach performs better than that of SN and SE approaches.

Key words: Parallel databases, shared-nothing, shared-everything, distributed shared memory

INTRODUCTION

The rise in the complexity of databases in terms of physical size, query complexity and query volume demands enormous amount of processing power which can only be satisfied using parallel systems (Pirahesh *et al*., 1990). Parallel system architectures range between two extremes, the shared-nothing and the shared-memory architectures. Both architectures have their pros and cons. The SN architecture is suitable to large scalable systems but load balancing requires complex data partitioning and assignment strategies. In SE architecture the global accessibility of data allows uniform utilization of resources. However, the uniformity of a SE system makes communication a bottleneck which hinders the scalability of such a system. Distributed Shared Memory (DSM) (Carter *et al*., 1991; Li and Hudak, 1989) can be used to overcome the drawbacks of both SE and SN. In a DSM system, memory is physically distributed among the processors. When data is partitioned among processors, a DSM system behaves as a SN system. On the other hand the uniform view of the memory allows idle processors to work on non-local data and better utilize the processing resources in the system. However, it is not just enough to have a scalable architecture to build a scalable database system with the desired features. The database system built top in the architecture should be able to utilize the architecture properly. In order to achieve this, we propose the scalable sharing (SS) approach.

The key to the SS approach is to recognize the ability of the DSM architecture to behave both as a SN system and a SE system. Thus, the database design should be able to handle the critical issues in both SN and SE systems. In SN systems it is data partitioning and in SE systems it is scheduling. In SS system design both issues become important and thus have to be designed to take advantage of the capabilities provided by each. In this paper we present the design of a SS system with respect to the underlying architecture, data partitioning and scheduling.

Our objective is to build a database system that captures the advantages of both SN and SE approaches. From the discussion in the literature of Stonebraker (1986) we know that the main drawback of sharing is scalability. We identify the following guidelines to build a scalable system:

- Eliminate central resources that can become bottlenecks
- Minimize communication and thus avoid making the interconnect a bottleneck
- Utilize modular, hierarchical coupling that will allow larger systems to be built from smaller subsystems

Our approach to satisfying these guidelines to scalability contains three components:

- The underlying hardware architecture
- Data partitioning
- Scheduling

Corresponding Author: Pushpa Rani Suri, Department of Computer Science and Applications, Kurukshetra University, Kurukshetra-136119, Haryana, India Tel: 09416352227

We will describe previous work related to each of these components and the innovations and extensions we propose.

ARCHITECTURE

Parallel system architectures range between two extremes, the Shared-Nothing (SN) and the Shared-Everything (SE) architectures. In the shared-nothing approach (Fig. 1), each processor has exclusive access to its main memory and disk unit(s). Thus, each node can be viewed as a local site (with its own database and software). In addition, shared-nothing architecture has three main virtues: cost, extensibility and availability. On the other hand, it suffers from higher complexity and (potential) load balancing problems.

Shared-nothing architectures minimize interference by minimizing resource sharing. They also exploit commodity processors and memory without needing an incredibly powerful interconnection network. As Fig. 1 suggests, the other architectures move large quantities of data through the interconnection network. The shared-nothing design moves only questions and answers through the network. Raw memory accesses and raw disk accesses are performed locally in a processor and only the filtered (reduced) data is passed to the client program. The SN architecture places the burden of ensuring on how the data is split and assigned to nodes.

Examples of shared-nothing parallel database systems include the Teradata's DBC and Tandem's non- stop SQL products as well as a number of prototypes such as BUBBA (Boral *et al*., 1990), GAMMA (DeWitt *et al*., 1990), PRISMA (Apers *et al*., 1992) and ARBRE (Lorie *et al*., 1989).

In the Shared-Everything (SE) approach (Fig. 2), any processor has access to any memory module or disk unit through a fast interconnect (e.g., a high-speed bus or a cross-bar switch). Several new mainframe designs such as the IBM3090 or Bull's DPS8 and symmetric multiprocessors such as Sequent and Encore, follow this approach. Shared-memory has two strong advantages: simplicity and load balancing. These are offset by three problems: cost, limited extensibility and low availability.

In a SE system the data repositories (memory and/or disks) and processors are independent of each other and are connected through an interconnection network. In a SE system all processing nodes are uniform i.e. a given task can be processed by any processor in the system. This uniformity allows a high degree of flexibility in scheduling tasks and amount of parallelism employed for transactions. However, the data required by a task has to be moved between the data repositories and the processors through the interconnect. This makes the interconnect a central resource and thus a bottleneck to scalability.

Examples of shared-everything parallel database systems include XPRS (Stonebraker *et al*., 1988), DBS3 (Bergsten *et al*., 1991) and Volcano (Graefe, 1990). All the shared-memory commercial products (e.g., Ingres and Oracle) today exploit inter-query parallelism only (i.e., no intra-query parallelism).

The SN and SE approaches represent the extremes of spectrum of possible approaches. We propose a

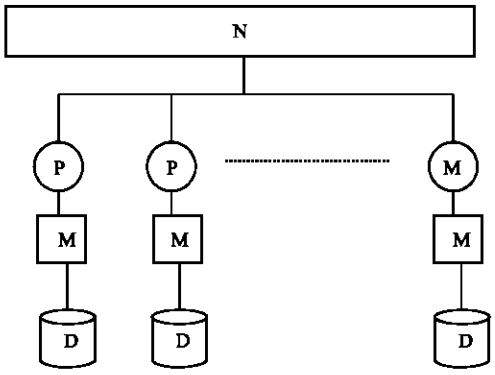

Fig. 1: Shared-nothing architecture

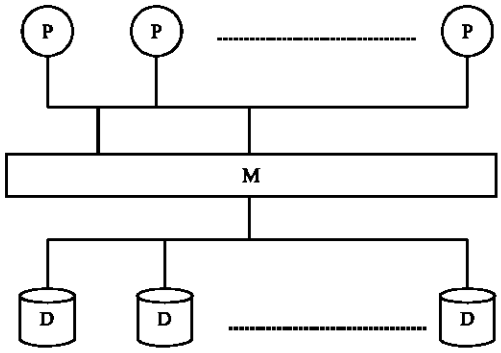

Fig. 2: Shared-everything architecture

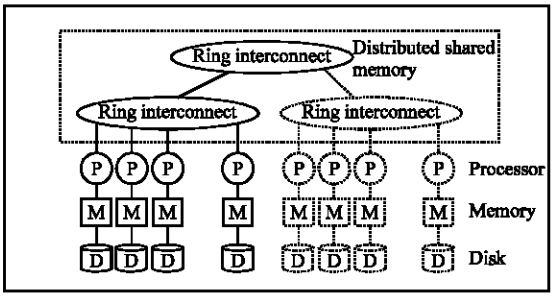

Fig. 3: A scalable sharing architecture

Distributed Shared Memory (DSM) based architecture that provides the flexibility of a SE system and the scalability of a SN system (Fig. 3). In Bellew *et al.* (1990), present a DSM based architecture. However, the thrust of their work was efficient coherence maintenance and concurrency control. DSM was also used in Shatdal and Naughton (1993) to enhance join strategies to efficiently handle data skew. However, these studies did not examine the impact of architectural assumptions on the scalability of the database system.

The DSM architecture we propose consists of interconnected processors, each with a large amount of main memory and a disk. The main difference between this architecture and SN architecture is that the interconnect supports a shared address-space thus allowing a processor to access data irrespective of its location. The DSM architecture differs from the SE architecture in that all nodes are not identical with respect to the cost of accessing a particular data item. The architecture we propose is an extension to existing scalable, shared-memory multiprocessor architectures. The architecture provides shared-memory. In hardware and not as a software abstraction built on top of a message-passing environment such as those in (Carter *et al.*, 1991; Li and Hudak, 1989). The reason for choosing a hardware implementation of DSM, despite the ease of implementation of the software based DSM, is the higher cost of sharing a software based DSM. The nodes in this architecture are interconnected through a ring. The architecture is scaled, not by adding more nodes to a single ring, but by hierarchically coupling such rings, i.e., rings of nodes are connected through a higher level ring (Fig. 3).

Before we discuss how to build a scalable database system on the proposed hardware architecture, we have to recognize that the SN mode of functioning is acceptable as long as all the nodes are busy processing assigned tasks that access data local to the nodes. However, the scheduling flexibility of a SE system is needed once the system has a mix of idle and busy nodes at which time both tasks and data may have to be moved from busy to idle nodes. We call this approach to sharing.

SCALABLE SHARING

Data partitioning: For the database system to be able to behave as a SN system we have to partition the data among the processors. Data partitioning (declustering) is a simple and effective way of achieving both inter and intra-query parallelism.

Data partitioning has two phases: Logical and physical partitioning. Logical partitioning divides a relation or a class into a collection of disjoint subsets of tuples or object respectively. Physical partitioning maps these subsets to processors.

Logical partitioning: A relation or a class can be declustered by applying a partitioning function (e.g., hash, range) to an attribute to decide the mapping between the tuple and a fragment (DeWitt and Gray, 1992; Boral *et al.*, 1990). However, when a single attribute is used for the declustering, only queries that involve the declustering attribute gain the full benefits of the available parallelism. This problem can be overcome by using multi-attribute declustering (Ghandeharizaeh *et al.*, 1992; Li *et al.*, 1992). First, all, the attributes to be used in the declustering are identified. Then the domain of values for each such attribute is divided into a set of sum-domains based on their attribute values. The number of sub-domains along each dimension is dependent on the degree of parallelism that is desired. Fro example the MAGIC declustering method (Ghandeharizaeh *et al.*, 1992) applies range partitioning to each of the attributes involved in the declustering to create a multi-attribute grid structure.

The desired degree of parallelism, p, can be determined when following cost model is assumed (Wilschut *et al.*, 1992; Ghandeharizaeh *et al.*, 1992):

$$R = C_t T_{ave}/P+P_{CP} \quad (1)$$

$$P_{opt} = \sqrt{(C_t T_{ave}/C_p)} \quad (2)$$

where, R is the response time for a query, C_t is the cost of processing a tuple, T_{ave} is the average number of tuples accessed by a query and C_p is the overhead per unit of parallelism. Then the degree of parallelism that gives least response time can be obtained by differentiating Eq. 1 to give Eq. 2. Thus Popt serves as a lower bound for the number of sub-domains in a dimension. In the MAGIC declustering strategy the number of ranges in a dimension can be greater than or equal to P_{opt}. In our approach we divide each dimension i into exactly P_{opt} ranges.

Physical partitioning: The result of logical partitioning is a per relation multi-dimensional grid of fragments, each containing a disjoint subset of the relation's tuples. Each fragment is made up of one or more pages. The physical partitioning strategy maps these fragments to the available nodes in the system with an

objective to guarantee the desired degree of parallelism along each dimension. The grid structure, together with the mapping information, form declustering directory. The simplest such mapping would be to assign each fragment in a relation to a different and repeat the process for each relation. However, the case where the number of fragments is greater than the number of processors this strategy will not work and a more complex assignment strategy is required. Ghandeharizaeh *et al.* (1992) present a set of complex heuristics to assign the fragments to processors.

We propose a simple algorithm, the diagonal assignment strategy, for assigning fragments to processors that does not make use of any heuristics. The diagonal assignment strategy assigns fragments to a subset of available nodes in the system. The diagonal assignment strategy groups fragments into slices and assigns each slice to a different node. This assignment strategy guarantees the same degree of parallelism for any query accessing a relation and requires at most as many processors as max(P_{opt}). For example, assume that the declustering is on two attributes and P_{opt} for both dimensions is the same (P), i.e., the result of logical partitioning is a two dimensional, P×P grid. The diagonal assignment strategy creates P slices, each with P fragments in it. If the fragments in the grid are numbered 1,2,……., P×P in a row major order then, the number if ith slice can be determined using the equation:

$$Slice_i = ((i-1)\,P + 1 + j\,(P+1)) \bmod n \quad (3)$$

$$\text{For all } \forall\, j \in [0, P);\ 1 \le i \le P;\ n = P \times P$$

Figure 4 shows the assignment of a 5 grid of fragments to nodes. The numbers in the boxes show the node to which a particular fragment is assigned.

As shown in Fig. 4, our assignment strategy uses only a subset of nodes in the system to assign the fragments of a given relation and fragments of different relation are assigned to different subsets of nodes in the system in a round-robin manner. We use a simple example to illustrate the difference between our strategy and the full-declustering strategy. Let us assume that there are 6 relations the database, each with a 3×3 grid (9 fragments).

Let us assume that the system has 9 nodes. Now, in a full- declustering strategy each fragment of each relation would be assigned to one processor which would result in 6 fragments per node (1 fragment/ relation×6 relations). Using our strategy, each relation's grid would be split across a subst of three nodes. Thus node 1-3 get relation 1 and 4, nodes 4-6 get relations 2 and 5 and nodes 7-9 get relations 3 and 6. once again each node gets 6 fragments

1	**3**	**3**
2	1	**2**
3	**2**	**1**

Fig. 4: Diagonal-assignment for a 3×3 grid

(3 fragments/relation×2 relations), the difference being that a given relation is not fragmented across all nodes, but only a subset of the available nodes. Thus, intuitively the performance of the subset assignment strategy should be very similar to that of full-declustering strategy. However, our strategy greatly simplifies the process of assigning fragments to nodes in the system

Another problem in physical partitioning is the presence of data skew. In SN systems the basic technique for skew avoidance is to tune the assignment of fragments to processors (Hua and Lee, 1991, Omiecinski and Lin, 1992) such that each node gets roughly the same number of fragments. These strategies rae static in nature. Our approach to handling data skew is not to handle it at query execution time (static) but to handle it at query execution time (dynamic), which is explained further.

In the case of high correlation between the partitioning attributes, i. e., when a majority of the data in fragments along a diagonal in the declustering grid, the diagonal assignment strategy could adversely affect the performance, despite the best scheduling efforts. To overcome this deficiency of the diagonal assignment strategy, we could use the magic squares strategy which guarantees that even diagonal elements get assigned to different processors (Zhiyong *et al.*, 1992). This property of the Magic Squares strategy allows it to trivially handle the presence of data skew due to correlation between attributes.

Scheduling: Scheduling strategies in SN and SE systems are known as data-affinity and load-balancing scheduling respectively. In SN systems, queries are decomposed into sub-queries based on the degree of declustering of the data being accessed. The sub-queries are scheduled to be processed on nodes that store the corresponding fragments. In SE systems, queries are decomposed into sub-queries based on the degree of parallelism deemed

optimal to execute the query. Since all processors are uniform in a SE system, the sub-queries can be executed on any processor.

However, none of these scheduling strategies make the best use of the database architecture presented earlier. The SN style of scheduling is susceptible to data skew whereas the SE style of scheduling fails to recognize the non-uniform costs of data access in our architecture.

Our approach to scheduling decomposes queries into sub-queries and is processor initiated. However, our approach takes data-affinity into consideration and thus is similar to the scheduling approach presented in (Markatos and LeBlanc, 1992).

In our scheduling approach each node has a private work queue, similar to a SN scheduling strategy. However, any node can access any other node's work queue through DSM. When a query is submitted to the DBMS, the declustering directory is consulted to determine the fragments that need to be accessed. Based on this information the query is decomposed into sub-queries each of which perform the required operation on a single fragment. Each sub-query is then placed as a work unit in the privet work queue of the node to which the corresponding fragment has been assigned during physical partitioning (Fig. 5). When a processor is done with processing a work unit, it attempts to get fresh work from its private work queue, i.e., the first priority of a processor is to complete processing work units that access data local to that node. When a processor cannot find any work assigned to it, there are two choices. One, it can sit idle until work is assigned to it in which case the system essentially behaves as a SN system. With the diagonal assignment strategy the response time would then be constrained by the largest work unit and the available resources would not be fully utilized. The second choice is to let the processor try and find work from another processor's queue. Processing a work unit

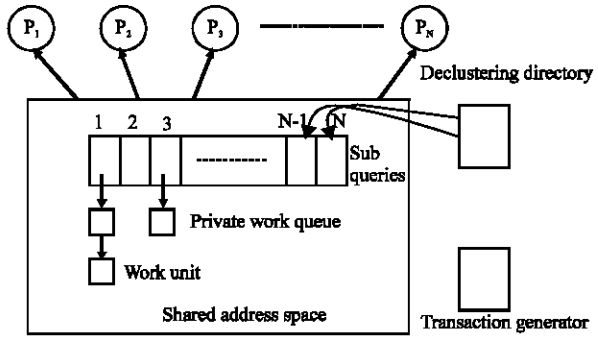

Fig. 5: The database model

assigned to a different processor involves greater processing time because of the data movement involved and has to taken into consideration in deciding where the work is to be obtained from.

Based on this frame-work for scheduling we can come up with several scheduling strategies. We briefly discuss one such strategy here. When a processor becomes idle and has no work units in its work queue, a work unit assigned to another processor is selected as follows:

- The work queue of the node with the most work pending is selected. The pending work is computed as the sum of number of pages that all the work units in that will access. We expect this strategy to reduce the load on the most loaded node because the time for processing a work unit is directly proportional to the number of pages that the work unit accesses
- The search space for a largest work queue is initially limited to those corresponding to the processor nearest to the idle processor. This is done because the cost of moving data between node pairs of different distances is different
- When a appropriate work queue is found, a non-modifying work unit is obtained and the data required to process the work unit is transparently moved through the DSM. The reason for choosing only non-modifying work unit is two fold. First, modifying transactions typically have no parallelism that can be exploited through load-balancing that justifies the cost of data movement. Second, no data coherence is required when data is moved only for non-modifying queries because such data can be discarded immediately after the node finishes processing the sub-query. We plan to use data reuse through caching in future

Here, we have presented the design of the components that are key to building a scalable system. The DSM architecture we have described has no central resources. The processor-disk pairs in combination with data partitioning, eliminate the I/O bottleneck. The data partitioning and data affinity scheduling together minimize the amount of communication required in processing the transactions while effectively handling load imbalances. The low communication requirement and the hierarchical interconnect remove the interconnection bottleneck. Also, the hierarchical architecture allows larger systems to be built by interconnecting rings of nodes in a hierarchical manner. Thus, we can see that the database architecture described here meets the guidelines to building scalable systems that we identified in the beginning.

REQUIREMENTS FOR PARALLEL DATABASE SYSTEMS

The criteria used in the comparison of architectures of parallel database systems rely on the following set of requirements (DeWitt and Gray, 1992; Stonebraker, 1986; Valduriez, 1993):

- Good scalability
- High data availability
- Efficient load balancing
- Low cost of interprocessor exchanges
- Low overheads on ensuring cache coherence
- Efficient organization of the concurrency control

Let us consider the specified criteria in more detail.

Scalability: System scaling is referred to as dynamic buildup a system to adapt to a growing database size or increasing performance requirements by gradually incorporating additional processors, memory modules, disks and other hardware components into the system. If the hardware capacity of the system doubles, its performance is expected to double as well. However, in practice, a real increase in the performance is usually much lower. For example, the scalability of the SE systems is limited to 20-30 processors (Valduriez, 1993). With the further enhancement of an SE system, its performance grows very slowly or even starts to fall (Martin *et al*., 1994). This is explained by the fact that the processors spend much time waiting for an access to the shared resources. Hence, the scalability of any multiprocessor system is determined by the parallelization efficiency.

The parallelization efficiency is described in terms of two basic qualitative characteristics: speedup and scaleup (DeWitt and Gray, 1992). The architecture of a multiprocessor system is considered to be nicely scalable if it demonstrates almost linear scaleup and speedup. Linear scaleup implies that the time spent by the system for solving a problem is equal to that spent by a double system for solving a double problem. Linear speedup implies that a double system solves a problem twice as fast as the original system. The main factor that worsens the scalability of systems results from drawbacks associated with the concurrent access of shared resources by the processors.

Data availability: One of the critical characteristics of parallel database systems is the capability of the system to ensure a high degree of data availability under the condition of failures of some hardware components. The probability of a hardware failure in a one-processor system is not great. However, in a system with thousands of processor nodes, this probability increases thousand fold. Therefore, the problem of ensuring high data availability in multiprocessor systems is of great importance.

Load balancing: The balancing of the processor load is one of the key problems in ensuring high efficiency of the parallel query processing. The DBMS should divide a query into parallel agents and distribute them among the processors to ensure uniform loading of all processors. The problem of load balancing is especially important in the case where partitioned parallelism is used (DeWitt and Gray, 1992). The important factor affecting the efficiency of the parallelization of database operations (especially, join and sort operations) is the value of the skew in data to be processed. It has been shown that, in real databases, some values of a certain attribute occur more frequently than others (Yannis *et al*., 1992; Hou and Kindred 1993). In particular, Lynch (Yannis *et al*., 1992) notes that the values of text attributes are usually distributed in accordance with the Zipf law. Such non-uniformity is said to be the attribute value skew (Annita *et al*., 1995). Lakshmi *et al*. (2000) showed that, in the presence of data skew, the speedup of the parallel execution of the join operation may be extremely low because of an overload of some processors and underload of others.

Interprocessor communications: If the partitioned parallelism is used, the interprocessor communications in parallel database systems can generate considerable traffic (Valduriez, 1993). This is explained by the fact that, upon parallel execution of the operation of joining 2 relations, we have either to dynamically fragment anew the original relations by the join attribute or to send the alien tuples from one processor node to another. Both actions are associated with sending considerable amounts of data through the communication network. Therefore, the cost of the interprocessor exchanges may critically affect the total system performance.

Cache coherence: When a common disk pool is shared by several processors, we face the so-called cache coherence problem (Rahm, 1993). The essence of this problem is as follows. After a transaction addresses a disk page, the image of this page remains for some time in the buffer associated with the given processor node. Hence, one processor node may revoke changes made by the other processor node. To avoid this, any time when the disk page is accessed, we need to check whether the image of this page is contained in the buffer pools

(caches) of other processor nodes and, if this takes place, coordinate changes produced in the caches of these processor nodes.

Concurrency control: Another series problem for database systems with shared disks is the support of the global lock table (Mohan and Narang, 1992). The locking is one of the basic methods used for ensuring ACID properties of the transactions. If different processor nodes work concurrently with the same database objects, they must have an access to the common (global) lock table. The support of such global lock table in multiprocessor systems without shared memory can be associated with great overheads (Mohan and Narang, 1992).

The comparative analysis of the SE, SD and SN architectures was done by Stonebraker and can be found in the classical work (Stonebraker, 1986). This analysis showed that, from the standpoint of scaleable high-performance database systems, the SN architecture is most preferable among these three architectures.

COMPARATIVE ANALYSIS OF ARCHITECTURES OF PARALLEL DATABASE SYSTEMS

Hear, we compare SN and SS parallel architectures of database systems with our proposed approach using the criteria formulated mentioned earlier and are graded on a four-point basis: 0 (Unsatisfactory), 1(Satisfactory), 2 (Good) and 3 (Excellent).

Scalability: The SN architecture is characterized by the good scalability (2 points). This is associated with the fact that, in the case of many processor nodes, the interprocessor communication network becomes a bottleneck (Rahm, 1993; Norman *et al*., 1996). The SS architectures demonstrate better scalability (3 points) owing to the fact that most of the communications occur inside the rings, thus unloading the interring network.

Data availability: The SN architecture is characterized again as a good one (2 points). This is explained by the fact that the backup copies in an SN system should be partitioned to many nodes (Hsiao and DeWitt, 1993) in order that to make the backup copy of a failed disk available in the parallel mode (otherwise, there may arise a serious disbalance in the loading). The support of the coherence of the partitioned backup copies requires certain overheads associated, first of all, with sending large amounts of data through the communication network. The SS architectures demonstrate better data availability (3 points) owing to the fact that all problems related to ensuring high data availability can efficiently be solved at the level of separate rings.

Load balance: Load balance for the SN architecture is a serious problem, since the SN systems are very sensitive to the data skew (Lakshmi and Yu, 1990). Therefore, the corresponding grade of the SN architecture is 0. The hierarchical SS architectures make it possible to get better load balance since the load is balanced at two-intering and intraring-levels. Accordingly, the SS architecture get 1 point. The best load balance among the considered architectures is achieved in SE clusters, since, in addition to the disks, the entire operative memory is available for all processors (Valduriez, 1993).

Interprocessor communication: The high cost of interprocessor communications is a weak point of the SN architecture (Stonebraker, 1986; Englert *et al*., 1995) (0 points). The SS architecture outperform the SN architecture in terms of this criterion since, potentially, the intraring communications can be implemented more efficiently than the interring communications (Sokolinsky, 1999). Accordingly, we give 1 point SS architecture.

Cache coherence: Cache coherence is a not a serious problem for both the SN and SS architectures. Therefore, the SN and SS architectures get the highest grade (3 points).

Concurrency control: Concurrency control is related to difficulties associated with the organization of the database object locking by the concurrent transactions accessing them. Therefore, the SN architecture is the best in terms of this parameter (3 points). The SS architecture fully inherits this feature from the SN architecture (also 3 points).

Conclusion: Based on the above analysis and taking into account the sum of the grades for different criteria shown in the Table 1, we may conclude that the SN architecture in the pure form is not appropriate. However, if we take into account the entire collection of the requirements to parallel database systems, we can see that the SS architecture is the best one.

Table 1: Comparison of architectures

	SN	SS
Scalability	2	3
Data availability	2	3
Load balancing	0	1
Interprocessor communications	0	1
Cache coherence	3	3
Concurrency control	3	3
Sum of points	10	14

CONCLUSION

In this research we have presented a new database architecture called SS, which offers the scalability of a SN system and the flexibility of a SE system using DSM for data movement. Further, on the basis of basic requirements to parallel database systems. we have carried out comparative analysis of SN and SS architectures of parallel database systems. This analysis has revealed that the SS architecture has the best performance.

REFERENCES

Annita, N.W., J. Flokstra and A.M.G. Peter, 1995. Parallel evaluation of multi-join queries. Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. May 1995, ACM Press, San Jose, California, pp: 115-126.

Apers, P., C. Van Den Berg, J. Flokstra, P. Grefen and M. Kersten *et al*., 1992. Prisma/DB: A parallel main-memory relational DBMS. IEEE. Trans. Knowledge Data Eng., 4: 541-554.

Bellew, M., M. Hsu and V. Tam, 1990. Update propagation in distributed memory hierarchy. Proceedings of the 6th International Conference on Data Engineering. IEEE. Computer Society. May 02-Sep. 02, Los Angeles, CA, USA, pp: 521-528.

Bergsten, B., M. Couprie and P. Valduriez, 1991. Prototyping DBS3, a shared-memory parallel database system. Proceedings of the International Conference on Parallel and Distributed Information Systems. April 12-June 12, Miami, Florida, IEEE Comput. Soc., pp: 226-234.

Boral, H., W. Alexander, L. Clay, G. Copeland and S. Sanforth *et al*., 1990. Prototyping bubba: A highly parallel database system. IEEE. Trans. Knowledge Data Eng., 2: 4-24.

Carter, J.B., J.K. Bennet and W. Zwaenepoel, 1991. Implementation and performance of munin. Proceedings of the 13th Symposium on Operating System Principles. Oct. 13-16, ACM Press, New York, NY, USA pp: 152-164.

DeWitt, D.J., S. Ghandeharizadeh, D.A. Schneider, A. Bricker and H.I. Hsiao *et al*., 1990. The GAMMA database machine project. IEEE. Trans. Knowledge Data Eng., 2: 44-62.

DeWitt, D.J. and J. Gray, 1992. Parallel database systems: The future of high-performance database systems. Commun. ACM., 35: 85-98.

Englert, S., R. Glasstone and W. Hasan, 1995. Parallelism and its price: A case study of nonstop SQL/MP. ACM SIGMOD Record, 24: 61-71.

Ghandeharizaeh, S., D.J. DeWitt and W. Qureshi, 1992. A performance analysis of alternative multi-attribute declustering strategies. Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data. June 02-05, ACM Press, New York, pp: 29-38.

Graefe, G., 1990. Encapsulation of parallelism in the volcano query processing systems. Proceedings of ACM SIGMOD International Conference Atlantic City. June 1990, ACM Press, NJ, USA, pp: 102-111.

Hou, W.C. and J. Kindred, 1993. Implementation and evaluation of relational algebra operations on the connection machine. Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems. January 20-22, San Diego, CA, USA, pp: 251-254.

Hsiao, H.I. and D.J. DeWitt, 1993. A performance study of three high availability data replication strategies. Distributed Parallel Databases, 1: 53-80.

Hua, K.A. and C. Lee, 1992. Handling data skew in multiprocessor database computer using partition tuning. Proceedings of the 17th International Conference on VLDB. 1992 Morgan Kaufmann, pp: 525-535.

Lakshmi, M.S., M. Seetha and P.S. Yu, 2000. Effect of skew on join performance in parallel architectures. Proceedings of the 1st International Symposium on Databases in Parallel and Distributed Systems. December 05-07, IEEE Computer Society Press Los Alamitos, CA, USA, pp: 107-120.

Li, J., J. Srivastava and D. Rotem, 1992. CDM: A multidimensional declustering method for database systems. Proceedings of the 18th VLDB Conference. August 23-27, Morgan Kaufmann, San Francisco, CA, USA, pp: 3-14.

Li, K. and P. Hudak, 1989. Memory coherence in shared virtual memory systems. ACM TOCS, 7: 321-359.

Lorie, R., J.J. Daudenarde, G. Hallmark, J. Stamos and H. Young, 1989. Adding intra-parallelism to an existing DBMS: Early experience. IEEE. Bull. Database Eng., 12: 2-8.

Markatos, E.P. and T. LeBlanc, 1992. Load balancing vs. locality management in shared-memory multiprocessors. Proceedings of the International Conference On Parallel Processing, 1992 CRC Press, pp: 258-267.

Martin, T.P., P.A. Larson and V. Deaspande, 1994. Parallel hash-based join algorithms for a shared-everything environment. IEEE Trans. Knowledge Data Eng., 6: 750-763.

Mohan, C. and I. Narang, 1992. Efficient locking and caching of data in the multisystem shared disks transaction environment. Lecture Notes Comput. Sci. Springer, 580: 453-468.

Norman, M.G., T. Zurek and P. Thanisch, 1996. Much Ado about Shared-Nothing. ACM SIGMOD Record, 25: 16-21.

Omiecinski, E. and E. Lin, 1992. The adaptive hash join algorithm for a hypercube multicomputer. IEEE. Trans. Parallel Distributed Syst., 3: 334-349.

Pirahesh, H., C. Mohan, J. Cheng, T.S. Liu and P. Selinger, 1990. Parallelism in relational database systems: Architectural issues and design approaches. Proceedings of the 2nd International Symposium on Databases in Parallel and Distributed Systems. July 2-4, ACM Press, pp: 4-29.

Rahm, E., 1993. Parallel query processing in shared disk database systems. ACM SIGMOD Record, 22: 32-37.

Shatdal, A. and J.F. Naughton, 1993. Using shared virtual memory for parallel join processing. Proceedings of the 1993 ACM SIGMOD. May 26-28, ACM Press, USA, pp: 119-128.

Sokolinsky, L.B., 1999. Operating system support for a parallel DBMS with a hierarchical shared-nothing architecture. Proceedings of the 3rd East European Conference Advances in Databases and Information Systems (ADBIS'99) (Maribor, Slovenia, 1999). September 13-16, Maribor: Institute of Informatics, Springer, pp: 38-45.

Stonebraker, M., 1986. The case for shared nothing. Database Eng. Bull., 9: 4-9.

Stonebraker, M., R. Katz, D. Patterson and J. Ousterhout, 1988. The Design of XPRS. Proceedings of 14th International Conference on Very Large Data Bases. September 1988, Morgan Kaufmann, Los Angeles, pp: 318-330.

Valduriez, P., 1993. Parallel database systems: The case for shared-something. Proceedings of the 9th International Conference on Data Engineering. April 19-23, IEEE Comput. Soc., Vienna, Austria pp: 460-465.

Wilschut, A.N., J. Flokstra and P.M.G. Apers, 1992. Parallelism in a main-memory DBMS: The performance of PRISMA/DB. Proceedings of the 18th VLDB Conference. August 1992, Morgan Kaufmann, pp: 521-532.

Yannis, E.I., T.N. Raymond, S. Kyuseok and K.S. Timos, 1992. Parametric query optimization. pp: 103-114

Zhiyong, L., L. Xiaobo and Y. Jia-Huai, 1992. On storage schemes for parallel array access. Proceedings of the 6th International Conference on Supercomputing. July 19-24, ACM Press, New York, USA, pp: 282-291.