Massive Data Management in Parallel Machines

Raymond Paul, M. Farrukh Khan, Ishfaq Ahmad, Omran Bukhres,
Imran Ghafoor, Amrit Goel and Arif Ghafoor

Abstract

In this paper we discuss issues related to the design and and usage of database management of massive amounts of data in parallel environments. The issues include the placement of the data in the memory, file systems, concurrent access to data, effects on query processing, the implications of specific machine architectures, and the peculiarities of specific parallel systems. Since not all parameters are currently amenable to rigorous analysis, results of performance studies are highlighted wherever deemed appropriate [1].

1 Introduction

Current real-world applications demand database size and processing capabilities beyond the capacity of the largest and fastest transaction processing systems available today. Since management, collection, processing and distribution of information has a major role in our lives, we can expect that future applications will demand even more computing power. Today, some of the leading such applications include: handling of scientific data from satellites and space missions, processing of the data for the human genome project, handling databases for national administration eg. social security, management of multimedia data, utilization of multidatabases spanning across corporations or other organizations, collecting data and performing simulations for studying changes in the global climate, data handling for weather forecasting and environmental studies, etc.

The advent of high-performance scalable parallel computers has provided a hope for handling some of the problems generated by these and other potentially massive databases. Despite the existence of the hardware, however, it is far from clear how the power of this breed of parallel machines can be harnessed to solve the problems at hand. To date, most of the work on parallel machines has focussed on producing efficient algorithms for the solution of computationally intensive problems. Very little work has coupled this emphasis on solution of computationally challenging problems with the concomitant, and sometimes contradictory, demands placed by data-intensive applications, such as those found in massive databases. In order to tackle the above problems we need a careful evaluation of the following factors, among others: hardware architecture, file and data structures, data layout and distribution schemes, and transaction processing models.

Figure 1 shows a layered organization of the issues mentioned above. Each of the higher levels in the layered organization depends on all the levels below it. The lowest level dealing with machine architecture issues determines what sort of file structures and file access methods are possible on a given processor interconnection, memory, and communication system. File organization influences the kind of data structures that can be built for manipulating data in these files. Algorithms are entwined with the data structures implementable on the underlying machine. Algorithm A may perform worse than algorithm B for a given architecture, but algorithm B may give better results than algorithm A on other machine architectures. Other factors that may influence the choice of data structures include the size of data set, reliability requirements, cost etc. In general, designers assume some kind of lower level layers when designing higher level layers, such as algorithms for concurrency control in database transaction processing systems. An example of such dependence is the design of main-memory systems which allow the optimization

[1] This research was supported in part by the National Science Foundation under grant number 8913133A1-ECD, awarded to School of Elect. Engr., Purdue University.
R. Paul is with OEC-US Army, Alexandria, VA, 22302.
M. F. Khan and O. Bukhres are with Dept. of Computer Sc., Purdue University, W. Lafayette, IN, 47907.
I. Ahmad is with Dept. of Comp. Sc., Hong Kong University of Science and Technology, Hong Kong.
I. Ghafoor is with Dept. of Business and Computing, Bronx Community College, City University of New York, The Bronx, New York.
A. Goel is with Dept. of Elect. and Comp. Engr., Syracuse University, Syracuse, New York, 13244.
A. Ghafoor is with School of Electrical Engr., Purdue University, W. Lafayette, IN, 47907.

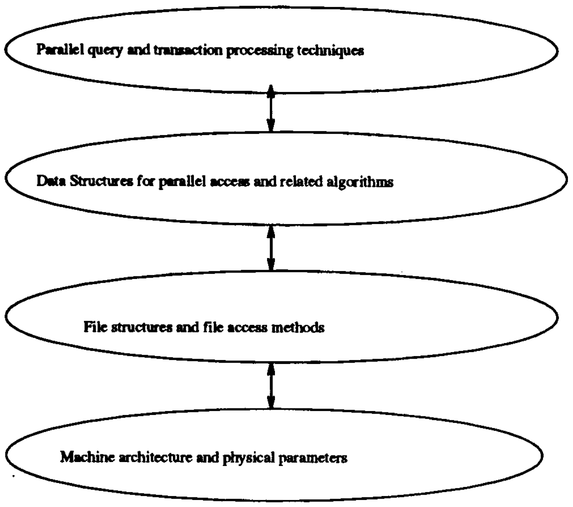
Parallel query and transaction processing techniques
Data Structures for parallel access and related algorithms
File structures and file access methods
Machine architecture and physical parameters

Figure 1: Layered Semi-dependencies of Different Sections of a Parallel Database Systems

of data-structures and algorithms to take advantage of the available main-memory and the lack of disk accesses. The layered organization is not very rigid. Often a machine may have some algorithms built in the hardware. An example of this is the parallel hardware sorter built described in [1].

The discussion in this paper follows the layered organization of Figure 1.

2 Architectural Issues in Parallel Databases

The architecture of the parallel computer is critical in the selection of data layout schemes and the kind of algorithms that can be employed to solve database problems. In particular, we are concerned about the issues of data placement and the performance of parallel algorithms available with the given data structures.

A number of considerations prevail in the construction of parallel architectures for database processing use. These are described in the following.

A choice between *general architectures* and *special purpose architectures* has to be made. Special purpose architectures are designed to serve a particular problem domain, such as image processing, real-time processing, or pure number crunching. When an architectures is tailored to solve database related-problems, the systems are also known as ***database machines***. Several database machines have been developed in order to handle massive amount of data or to have a high transaction processing rates. There can be various types of parallelization that influences the design, such as inter-query, intra-query and intra-operation parallelization.

Granularity, which is a measure of the amount of computation in a software process, is another major design parameter for parallel machines. Parallel machines may be looked upon as a collection of processor–memory pairs. These machines can either be ***coarse–garined***, ***fine–grained***, or ***medium–grained.***

Scalability of a parallel system determines its performance when the number of processors used for a given application is increased, or for a fixed number of processors, the problem size is increased. Scalability determines the matching between a computer architecture and a give application. For different combinations of problem and machines size, the performance analysis may be different. Therefore, for very large database application, scalability measure is an important factor for determining the suitability of an architecture.

Interconnection topology is another design issue in parallel architectures. A detailed treatment of how different topologies influence the design of parallel algorithms for those topologies is given in [2]. Static networks are usually employed in distributed-memory message passing The interconnection network affects the performance of communication primitives. In database applications, it is important to analyze the patterns of communication and then choose an interconnection network accordingly.

Memory Sharing refers to the way the programmer views the memory of the system. A shared memory system provides a single address space and a single coherent memory. Here, communication is done implicitly by directly accessing a common memory. In a non-shared memory system, on the other hand, each processor has its own local memory. A non-shared memory system, which employs distributed memory, can be viewed as a collection of processor memory pairs where Communication is done using explicit message-passing. Shared-memory system can employ either centralized memory or a distributed memory. Examples of distributed shared memory include the DASH and KSR systems [3] and Cray T3D. Since the access to the memory (read/write) can be the major factor affecting the performance of a parallel database, this is extremely important consideration for database applications and is elaborated further later.

Control mechanisms is important architectural component. Control mechanism determines the timing and synchronization of execution of instructions of the programs. Based on this, parallel machines are conventionally classified as SIMD (single-instruction, multiple-data), or MIMD (multiple-instruction, multiple-data).

Examples of SIMD computers include CM-2, MasPar MP-1 nd MP-2, and DAP610. Examples of MIMD computers include Thinking Machines CM-5, Intel Paragon, KSR-1, nCube, etc. Parallel database systems have been developed on both SIMD and MIMD architectures [5].

Mapping of tasks and data within a machine greatly impacts the performance of the machine [11, 7]. Mapping deals with the distribution of different tasks generated by an algorithm among the nodes of the system.

I/O Capabilities is another crucial factor for parallel processors. Current disks are very slow as compared to the CPU and the main memory. One proposed solution is to keep the entire database in the main memory [8]. However, this is a very expensive solution and precludes its use for all but the smallest databases. Also the trend has been that CPU and main memory bandwidth has been increasing much faster than disk memory bandwidth. Another solution is parallelization of I/O has been proposed as the solution to this problem [10, 9]. This can be achieved using disk arrays where the data is not placed on just a few high density disks. Rather, it is distributed over a large number of disks. This allows parallel access to different segments of data, thus increasing the effective bandwidth of I/O. In order to alleviate the burden of I/O from the main processors, the use of special I/O processors is a common practice in parallel processors. For example, the Intel Paragon [6], which is an MIMD machine based on a 2D mesh topology, has arrays of I/O processors on the right and left edges of the mesh. Similarly, the Thinking Machines CM-5 which is also an MIMD machine provides dedicated I/O processor. The number of I/O processors can be scaled with increasing number of processing nodes.

From memory usage point of view, architecture can be classified as *main memory databases*, *shared-memory architectures* and *I/O shared disk architectures*. These architectures have their pros and cons in terms of performance and database applications.

3 Data Structures and Data Organizations for Parallel Databases

Some important data structures that have been have been employed in the efficient access of data include, lists, and hashing. From parallel processing point of view, a number of concurrent algorithms for searching have been proposed for unbalanced and balanced trees. The usual mechanism to enforce non-interference between different threads of execution has been the use of locks. Concurrent algorithms for rebalancing or deletion from B-trees are generally very complicated.

The simulations results have revealed that skip lists show almost linear speedup with the increase in the number of concurrent threads of execution. The number of locks blocked is proportional to the number of locks held, which is proportional to the ratio of concurrent writers to the elements in the data structure.

With these results, it seems that skip lists provide efficient concurrent algorithms for use in parallel databases. The algorithms tend to be simpler than the corresponding ones using B-trees.

In the implementation of hash tables, it is required to implement the operations of inserting, deleting, and searching keys. Usually the hash table is stored in a manner that makes the distribution of data even over the entire system. If possible, each data element is associated with a single processor. Linear probing is used for resolving collisions, since this reduces communication costs in the system. The performance of the hash table should take into account the variance in the type of load that a system may be subjected to, as well as the communication overheads. These factors make the performance analysis of parallel hash systems much harder than conventional systems. [13] provides simple analysis of parallel hashed data system. The simulation study in [13] shows that performance of hash table with linear probing when the hash table is fully loaded is much worse than the performance of the hash table with 80 percent load when the only operation considered is *insert*. *Search* also has performance similar to insert.

4 File Manipulation Strategies

It is desirable to provide fast access and high bandwidth between the data stored on disks and the main memories associated with the processors. Files can be allocated as either *fixed blocks*, where all blocks are of the same size (eg. UNIX), or *extent based systems*, where allocation of data is as a few large and variable chunks of disk space. Fixed block systems have the disadvantage of discontiguous allocation of data on disk, and an excessive amount of book-keeping data. Extent based systems may thus provide higher performance. Performance studies which indicate the superiority of certain allocation policies are reported in [14]. In particular, striping across disks with contiguous allocation showed 250% improvement over policies without disk striping and contiguous allocation.

If the nature of data is non-static, as is the case when large number of updates are performed, or the nature of the system is unknown, provision should be made for even distribution of data (load-balancing) over the entire system via data reorganization. This load-balancing should not entail recompiling of the programs running on the system. This can be done by having associative access to the data in question.

A global index is kept indicating the placement of relations on the nodes. The index structure can be either B-tree based, or hash based. B-trees take more space, but range queries are more efficient. The global index is replicated on each node, so this may cause problems in scaling, because of consequent overhead [15].

Signature files are useful for associative retrieval on formatted or unformatted data files [16]. The major advantages of signature files over some other structures such as grid files or multi-dimensional hash structures are that the associative searches may be conducted over a large number of dimensions and this number may even vary for different records within the same file. Auxiliary files, called *signature files*, contain database record abstractions called *signatures*. Extensions to signature files in order to increase the performance of signature file query processing are possible.

Clustered surrogate files [17] are also used as an indexing scheme through a special data word, called the concatenated code word, or *CCW* for short. These CCW's constitute a *surrogate file*, which is small in size and simple to maintain through a small number of core operations. Considerable savings of time may be realized by performing related operations on the CCW surrogate files *before* performing them on the actual data files, which are often very large. Since the structure of surrogate files is compact and regular, mapping these files to different parallel architectures is not very complex.

5 Parallel Query Processing

Parallel query processing is an essential part of constructing any parallel database system, and can account for important performance improvements. In this section we look at work in parallel algorithms for database query processing. Parallelization of query processing provides opportunities for ***inter-query*** parallelism, ***intra-query*** parallelism, as well as ***intra-operation*** parallelism.

In ***inter-query*** parallelism, different queries are executed in parallel on different processors. ***Intra-query*** parallelism involves the parallel execution of different sub-operations within the same query. ***Intra-operation*** parallelism refers to the even more fine-grained parallelism, where single operations within queries are distributed over more than one processor for concurrent execution.

We assume that the operations to be performed on partitioned data in a parallel database consist of the basic relational algebra operators, or their derivatives. These include selection, projection, union, set difference, cartesian product, intersection and various kinds of join operations performed on the database relations. Researchers have generally concentrated on select and join operations, since these are basic and heavily used primitives in database query processing.

Divide and conquer strategy is applied to break up operations with a large number of tuples into smaller chunks, assigning these chunks to different processors, and processing the chunks in parallel. For example, the sorting phase may be distributed over nodes such that there is relatively low amount of of data skew and computational load is also spread over the nodes. Without such load-balancing, the speedup achieved is limited due to under-utilization of the resources, and extra overheads, such as intermediate disk saves etc.

An algorithm to increase the amount of parallelism in the hash-join algorithm by using ***pipelining*** is proposed in [18]. The hash-table for both relations A and B is formed. Whenever a tuple is produced from either relation, it is hashed to find the hash-key. Tuples in the other relation with the corresponding hash-key are compared with the new tuple. Any matching tuples are sent to the output stream. If one of the relations is exhausted, the tuples from the other relation are no longer inserted in the hash-table. This is so because only the first hash table is used in processing the join from now on i.e. it becomes like the non-pipelined version of the hash-join algorithm. Besides pipelining this algorithm also offers the advantage of being *symmetric* with respect to its operands. This eliminates the need to compare and order the operands before inputting them to the hash-join algorithm.

Most algorithms to select the best strategy in performing joins involving more than one relation first form an intermediate representation, called the *join-tree*. Optimizations are performed on the tree before feeding the tree-nodes to the join algorithms. This is done so in GAMMA [19], PRISMA [20, 18] and other parallel database systems.

One may not always have the choice to select the shape of the tree, and the edges may have different costs, affecting the tree that is ultimately selected for performing the joins. GAMMA prefers linear trees with minimal total processing costs; [21] chooses to minimize the processing time on the longest path in the tree. The PRISMA database system combines the choice of tree with pipelining hash-join and distributing expensive operations over more processor nodes.

Algorithms have been proposed that perform joins relatively well even in the presence of data skew. One such algorithm is described in [22]. The output of the sort phase of the sort-merge algorithm is preprocessed before the join/merge phase. The largest skew elements are identified and are assigned to an optimal number of processors. This helps in load-balancing for the join phase. The algorithm is also reported to be *robust* with respect to data skew and the number of processors.

6 Performance Issues

In this section, we describe some of the fundamental issues that need to addressed for evaluating the performance of parallel database systems. The major goals of a parallel system are to obtain linear speedup and scalability. The speedup refers to the ratio of the execution time of a task on a serial machine to the execution time of the same task on a parallel machine. A linear speedup means that this ratio is N if the system is made N times larger. Speedup essentially measures how fast a fixed-size task is executed if the size of the system is increased. Scalability on the other hand measures how a system performs if both the size of the task and system is increased. Ideally, one would like to to execute an N-times larger task on an N-times larger system with the same time as the original task on the original system. In parallel databases, batch speedup is one of the most common measure. The batch speedup of a system measures improvements in the response times of large query processing as the number of processing nodes is increased to perform parallel query processing. This speedup is obtained by exploiting though multiprocessing the parallelism inherent within a complex query. Scalability of parallel databases is measured in two ways. The first way is to measure the increase in transaction processing throughput as the number of processing nodes and the size of database is increased. Since each transaction is usually a small task without much parallelism, enhancement in throughput is obtained by scheduling transactions to different processing nodes. The second way of measuring scalability is to study how the response time of large batch programs, such as large decision-support queries, as the processing nodes and database size is increased. This is obtained by using more processing nodes to process a complex query, without increasing the workload of a single processing node.

In reality, linear speedups and scalability are difficult to obtain due to a number of reasons. These include the inter-processor communication overhead, synchronization, initiation and termination of parallel tasks, etc.

7 Conclusions

It is clear from the foregoing sections that there are a large number of factors that warrant careful attention when designing parallel database systems for very large amounts of data. Among them are: the physical architecture of processor, memory, disks and the interconnection network; physical and logical layout and distribution of data, and the file system employed; the kinds of data structures used in accessing and processing the data; selection of algorithms for the performance on parallel machines of traditional database operations such as sorting, performing joins, projections etc.

Architectural decisions affect the amount of different kinds of parallelism that can be obtained in shared-disk or shared-nothing systems. Shared-disk systems provide advantages of load balancing and easier management. Shared-nothing systems provide the benefits of scalability and lower communication costs. Potential for bottlenecks is also reduced in shared-nothing systems.

Further issues that need to be considered include the routine operations such as reload, unload, and reorganization issues; heterogeneity issues; performance measurements for a mix of complex query work loads, and the relative advantages of parallel synchronous pipelining versus parallel asynchronous pipelining in the processing of database queries. Most of these warrant further research before definitive opinions are reached on them.

References

[1] Browne, J. C., Dale, A. G., Leung, C., and Jenevein, R., "Parallel Multi-stage I/O Architecture with Self-Managing Disk Cache for Database Management Applications", in *Proceedings of Database Machines: Fourth International Workshop*, Bahamas, March 1985.

[2] Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, 1992.

[3] Bell, G., "Ultracomputers, A Teraflop Before its Time", *Communication of the ACM*, August 1992, pp. 27-47.

[4] Baru, C. K., Frieder, O., Kandlur, D., and Segal, M., "Join on a Cube: Analysis, Simulation and Implementation", in *Database Machines and Knowledge Base Machines*, Kitsuregawa, M. and Tanaka, H. (editors), Kluwer, 1987.

[5] Frieder, O., "Multiprocessor Algorithms for Relational-Database Operators on Hypercube Systems", *IEEE Computer*, 13-28, Nov. 1990.

[6] Intel, *Paragon XP/S Product Overview* Supercomputer System s Division, Intel Corporation, Beaverton, OR, 1991.

[7] Bowen, N., Nikolaou, C., and Ghafoor, A., "On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computing Systems", *IEEE Transactions on Computers, 41*, 3, 257-273, March 1992.

[8] Leland, M. D. P., and Roome, W. D., "The Silicon Database Machine", in *The Proceedings of the 4th International Workshop on Database Machines*, Bahamas, March 1985.

[9] Patterson, D., Gibson, G., and Katz, R., "A Case for Redundant Arrays of Inexpensive Disks (RAID)", in *Proceedings of the ACM SIGMOD Conference*, 109–116, Chicago, June 1988.

[10] Kim, M., "Synchronized Disk Interleaving", *IEEE Tran. on Comp. C-35*, 11, November 1986.

[11] Ahmad, I., Ghafoor, A., "Semi Distributed Load Balancing for Massively Parallel Multicomputer Systems", *IEEE Transactions on Software Engineering, 17*, 10, 987–1006, October 1991.

[12] Yen, I., Leu, D., and Bastani, F., "Hash Table and Sorted Array: A Case Study of Multi-Entry Data Structures in Massively Parallel Systems", in *Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation*, College Park, MD, October 1990.

[13] Seltzer, M., Stonebraker, M., *Read Optimized File System Designs: A Performance Evaluation*, in *Proceedings of the IEEE 7th International Conference on Data Engineering*, 1991.

[14] Khoshafian, S., and Valduriez, P., "Parallel Query Processing of Complex Objects", in *Proc. of the Fourth Int. Con. on Data Engr.*, Los Angeles, CA, February, 1988.

[15] Faloutsos, C. and Christodoulakis, S., "Description and Performance Analysis of Signature File Methods for Office Filing", *ACM Transactions on Office Information Systems, 5*, 3, July 1987.

[16] Chung, S. M., "Parallel Relational Operations Based on Clustered Surrogate Files", in *Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation*, College Park, MD, October 1990.

[17] Wilschut, A., and Apers, P., "Pipelining in Query Execution", in *Proceedings of the PARBASE-90 Conference*, Miami, FL, March 1990.

[18] DeWitt, D., Gerber, R. H., Graefe, G., Heytens, M. L., Kumar, K. B., and Muralikrishna, M., "GAMMA — A High Performance Dataflow Database Machine", in *Proceedings of the 12th International Conference on Very Large Databases*, Kyoto, Japan, August 1986.

[19] Apers, P., Hertzberger, B., Hulshof, B., Oerlemas, H., and Kersten, M., "PRISMA, a Platform for Experiments with Parallelism", in *Parallel Database Systems*, Pierre America (Ed.), Springer-Verlag, 1990.

[20] Bodorik, P. and Riordon, J. S., "Heuristic Algorithms for Distributed Query Processing", in *Proceedings of the First International Symposium on Databases in Parallel and Distributed Systems*, Austin, TX, December 1988.

[21] Wolf, J., Dias, D., and Turek, J., "An Effective Algorithm for Parallelizing Hash Joins in the presence of Data Skew", *Proceedings IEEE 7th International Conference on Data Engineering*, 1991.

[22] DeWitt, D., et. al, "The GAMMA Database Machine Project", *IEEE Tran. on Knowlwdge and Data Engineering 2*, 1, 44–62, March 1990.

[23] DeWitt, D., and Gray, Jim, "Parallel Database Systems: The Future of High Performance Database Systems", *Communications of the ACM*, June 1992.

[24] Driscoll, J., Gabow, H., Sharairman, R., and Tarjan, R., "Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Computation", *Communications of the ACM 31*, 11, 1343–1354, November 1988.

[25] Ellis, C. S., "Concurrency in Linear Hashing", *ACM Transactions on Database Systems 12*, 2, 195–217, June 1987.