Journal of Theoretical and Applied Information Technology

www.jatit.org

MECHANISMS FOR PARALLEL QUERY EXECUTION

[1] PUSHPA R. SURI, [2] SUDESH RANI

[1] Reader, Department of Computer Sc. & Applications, K. U., Kurukshetra , India-1361191

[2] Ph.D. Student , Department of Computer Sc. & Applications, K. U. , Kurukshetra , India-136119

E-mail: [1]pushpa.suri@yahoo.com, [2]dhillon_sudesh@rediffmail.com

ABSTRACT

Parallel query processing has attained a lot of attention in the database community because customer applications on databases are growing in size and complexity with an ever increasing demand for performance. The main objective of parallel query processing is to achieve speedup, scaleup and high throughput. Earlier bracket model was used for parallelization. In case of bracket model it is not possible to possible to execute two operators in one process without resorting to some thread or coroutine facility. Also bracket model involves expensive interprocess communication system calls even in the cases when an entire query is evaluated on a single CPU. To overcome all these difficulties of bracket model we have developed a new model in which a single operator is used for parallelization. As compared to bracket model, all issues of control are localized in one operator (parallelizing operator) in this model. Also we can execute a complex query in a single or with a number of processes by using one or more parallelism operators in the query evaluation plan.

Keywords: *Parallel query processing, interoperation parallelism, intraoperation parallelism, Bracket model, Operator model.*

1. INTRODUCTION

Timely information has become increasingly important for today's competitive world. The number of information services or business relying on timely information is growing fast. It is reported that the data volume is increasing by 25-35 % per year, while at the same time the amount of data stored per person is increasing. Moreover, the massive amount of data is not only used for simple data intensive applications, but it is also used to extract information by relating the data stored. These demands foreseen cannot be easily met using traditional disk-based data-base technology, because the I/O bottleneck forms a physical limitation to improve responsiveness. Disk technology has shown an improvement of only a factor of 2 over last 10 years in response time and throughput. In contrast the CPU speed has been doubling every year [8]. These developments have led to research in parallel data-base systems consisting of a large number of off-the shelf, and therefore, cheap processors. The processors are interconnected by a high speed network. Typically each processor is equipped with a large amount of memory and a disk. By declustering the data over the available processors data can be accessed in parallel, leading to an improved response time [15].

In order to provide real-time responses to complex queries involving large volumes of data, it has become necessary to exploit parallelism in query processing. All high-performance computers today employ some form of parallelism in their processing hardware. It seems obvious that software written to manage large data volumes ought to be able to exploit parallel execution capabilities [3].

The goal of parallel systems is to obtain speedup and scaleup [4]. Speedup considers additional hardware resources for a constant problem size; linear speedup is considered optimal. In other words, N times as many resources should solve a constant-size problem in l/N of the time.

An alternative measure for a parallel system's design and implementation is scaleup, in which the problem size is altered with the resources. Linear scaleup is achieved when N times as many resources can solve a problem with 1/N times as much data in the same amount of time.

2. QUERY PROCESSING IN PARALLEL DATABASE SYSTEMS

Conventional query processing assumes a uniprocessor environment and query plans are executed sequentially. A query plan for a uniprocessor environment is called a sequential plan. Each sequential plan basically specifies a partial order for the operations. We call a query plan for a parallel environment a parallel plan. If the parallel plan satisfies the same partial order of operations as a sequential plan, it is called a parallelization of the sequential plan and each sequential plan may have many different parallelization. Parallelization can be characterized in the following three aspects :

Forms of Parallelism
We can exploit parallelism within each operators i.e. intra-operation parallelism and parallelism between different operators i.e. inter-operation parallelism. Intra-operation parallelism is achieved by partitioning data among multiple processors and having those processors execute this same operation in parallel. Inter-operation parallelism can be achieved either by executing independent operations in parallel or executing consecutive operations in a pipeline.

Units of Parallelism
Unit of Parallelism refers to the group of operations that is assigned to the same processor for execution.

Degree of Parallelism
Degree of Parallelism is the number of processes that are used to execute a plan fragment. In theory, the degree of parallelism can be greater than the number of available processors.

3. IMPLEMENTATION STRATEGIES

The purpose of the query execution engine is to provide mechanisms for query execution from which the query optimizer can choose - the same applies for the means and mechanisms for parallel execution. The general approach to parallelize a query execution engine is bracket model. The bracket model has been used in distributed-memory systems such as Bubba and Gamma [2, 5].

In the bracket model, there is a generic process template that can receive and send data and can execute exactly one operator at any point of time. A schematic diagram of a template process is shown in Figure1, together with two possible operators, aggregation and join. In order to execute a specific operator, e.g., a scan, the code that makes up the generic template "loads" the operator into its place (by switching to this operator's code) and initiates the operator which then controls execution; network I/O on the receiving and sending sides is performed as a service to the operator on its request and initiation and is implemented as procedures to be called by the operator. The

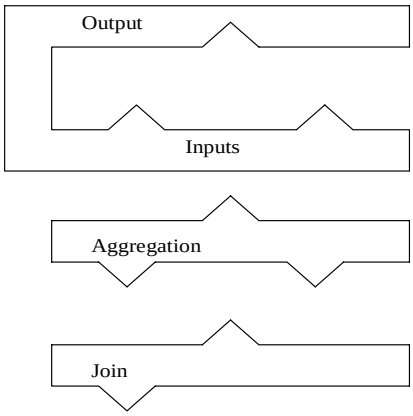

Figure1. Bracket model of parallelization.

number of inputs that can be active at any point of time is limited to two since there are only unary and binary operators in most database systems. The operator is surrounded by generic template code, which shields it from its environment, for example, the operator that produce its input and consume its output. For parallel query execution, many templates are executed concurrently in the system, using one process per template. One problem with the bracket model is that because each operator is written with the implicit assumption that this operator controls all activities in its process, it is not possible to execute two operators in one process without resorting to some thread or coroutine facility. Second problem in a query-processing system using the bracket model, that passing a data item from one operator to another always involves expensive interprocess communication system calls, even in the cases when an entire query is evaluated on a single CPU or when data do not need to be repartitioned among nodes in a network. The reason is that each operator is its own locus of control, and network flow control must be used to coordinate multiple operators, e.g., to match two operators' speed in a producer-consumer relationship.

To overcome the limitations of bracket model we are introducing an alternative to the bracket model called operator model. In this model, all issues of control are localized in one operator that uses and provides the standard iterator interface to the operators above and below in a query tree. Figure 2 shows a simple computatioin that combines temperature and pressure data for some further calculation such as digital filtering and display. Figure 3 shows a possible parallelization of this computation using the operator model, i.e., by inserting parallelism operators into a sequential plan. The parallelism operator is an iterator like all other operators in the system with open, next, and close procedures; therefore, the other operators are entirely unaffected by the presence of parallelism operator in a query evaluation plan. The parallelism operator does not contribute to data manipulation; thus, on the logical level, it is a "no-op" that has no place in a logical query algebra such as the relational algebra. On the physical level of algorithms and processes, however, it provides control not provided by any of the normal operators, i.e., process management, data redistribution, and flow control. Therefore, it is a control operator or a meta-operator. Separation of data manipulation from process control and interprocess communication can be considered an important advantage of the operator model of parallel query processing, because it permits design, implementation, and execution of new data manipulation algorithms such as N-ary hybrid hash join without regard to the execution environment.

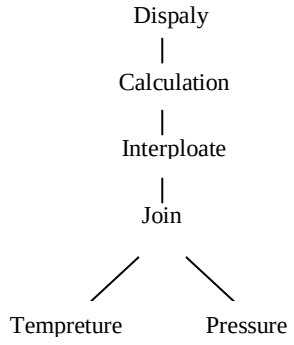

Figure2: Simple specific computation

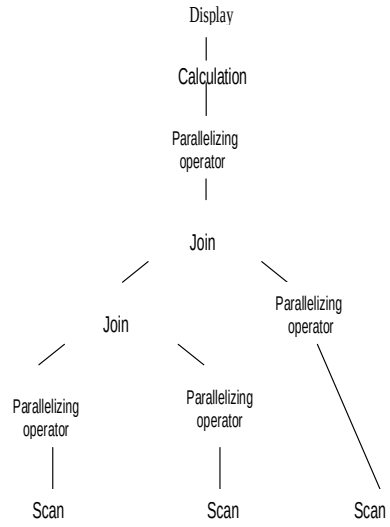

Figure 3: Operator model of parallelization

A second issue important to point out is that the parallelism operator only provides mechanisms for parallel query processing; it does not determine or presuppose policies for using its mechanisms. Policies for parallel processing such as the degree of parallelism, partitioning functions, and allocation of processes to processors can be set either by a query optimizer or by a human experimenter. The design of the parallelism operator permits execution of a complex query in a single or with a number of processes by using one or more parallelism operators in the query evaluation plan. The mapping of a sequential plan to a parallel plan by inserting parallelism operator permits one process per operator as well as multiple processes for one operator (using data partitioning) or multiple operators per process, which is useful for executing a complex query plan with a moderate number of processes. Earlier parallel query execution engines did not provide this degree of flexibility; the bracket model used in the Gamma design, for example, requires a separate process for each operator [5].Figure 4 shows the processes created by the parallelizing operator in the previous figure, with each circle representing a process. Note that this set of processes is only one possible parallelization. Furthermore, the degrees of data parallelism, i.e., the number of processes in each process group, can be controlled using an argument to the parallelizing operator.

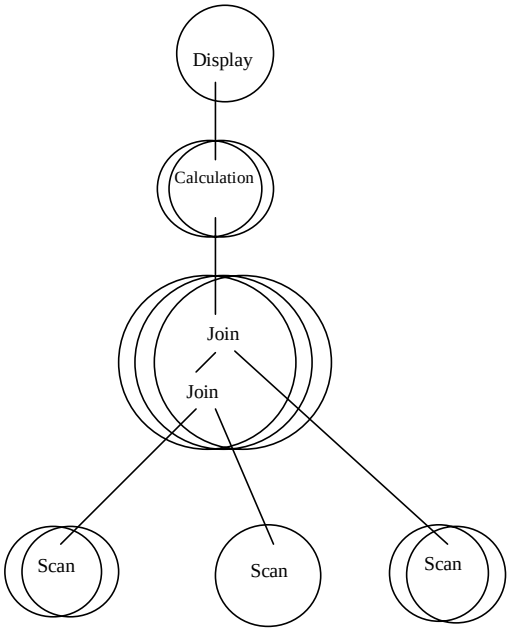

Figure 4. Processes created by Parallelizing operator

The difference with respect to performance is that the operator model permits multiple data manipulation operators such as join in a single process, i.e., operator synchronization and data transfer between operators with a single procedure call without operating system involvement. The important advantages of the operator model are that it permits easy parallelization of an existing sequential system as well as development and maintenance of operators and algorithms in a familiar and relatively simple single-process environment [10].

4. LOAD BALANCING AND SKEW

For optimal speedup and scaleup, pieces of the processing load must be assigned carefully to individual processors and disks to ensure equal completion times for all pieces. In interoperator parallelism, operators must be grouped to ensure that no one processor becomes the bottleneck for an entire pipeline. Balanced processing loads are very hard to achieve because intermediate set sizes cannot be anticipated with accuracy and certainty in database query optimization. Thus, no existing or proposed query processing engine relies solely on interoperator parallelism. In intraoperator parallelism, data sets must be partitioned such that the processing load is nearly equal for each processor. Notice that in particular for binary operations such as join, equal processing loads can be different from equal-sized partitions. There are several research efforts developing techniques to avoid skew or to limit the effects of skew in parallel query processing, e.g. [1,7,9,11,13,14,17,18,19,21]. However, all of these methods have their drawbacks, for example, additional equirements for local processing to determine quantiles.

Skew management methods can be divided into basically two groups. First, skew avoidance methods rely on determining suitable partitioning rules before data is exchanged between processing nodes or processes. For range partitioning, quantiles can be determined or estimated from sampling the data set to be partitioned, from catalog data, e.g., histograms, or from a preprocessing step.

Second, skew resolution repartitions some or all of the data if an initial partitioning has resulted in skewed loads. Repartitioning is relatively easy in shared-memory machines, but can also be done in distributed-memory architectures, albeit at the expense of more network activity. Skew resolution can be based on rehashig in hash partitioning or on quantile adjustment in range partitioning.

5. PARALLEL DATABASE ARCHITECTURES

Historically, the database community distinguished three types of system architecture as platforms for parallel database systems :

Shared nothing (SN): In this architecture, as the name indicates, processors have their own individual memory as well as disk space and do not share anything. Data coherency control is not a problem in a SN system. Processor and memory are physically localized in a node, and memory-access latency is not a problem. However, SN systems are very sensitive to data skew problems. When the data are seriously skewed, re-balancing the data load among processing nodes is necessary to resume good system performance.

Shared disk (SD): In this architecture, the processors have their own individual memory, but share disk space. Just like in SE, in SD, inter-processor coherency control is necessitated due to the caching of the shared database pages in main memory database buffers. The buffer invalidation problem tends to limit the size of a SD system.

Shared everything: In this architecture, processors share a single global memory address space. The shared memory is typically physically distributed memory to accommodate the aggregate demand on

the shared memory from a large number of processors. An interconnection network is usually used to allow any processor to access any memory module. This communication network increases memory-access latency, which effects the performance of conventional processors. To remedy this, a popular solution is to have cache memory with each processor. This again leads to cache coherency problems. Cache coherency control again requires a lot of interrogation and hence uses of the communication network. Hence Shared Everything model does not scale that well.

Since shared disk and SMP architectures are very similar in many respects, the comparison between shared nothing systems (mostly referred to as "massively parallel processing" (MPP-architecture)) on one hand and SMP systems on the other hand represent the architectural alternative for modern parallel database systems. Let us briefly compare them from the perspective of handling parallel database operators.

The properties of MPP-architectures can be characterized as follows:

- Such systems scale to hundreds and, maybe, thousands of processors.
- No data are shared among processors.
- Data living on different processors must be combined using messages.
- Data in the database are partitioned statically.

The properties of SMP-architectures can be characterized as follows:

- Such systems scale to tens, maybe hundreds of processors.
- Data are shared among all processors.
- Data are combined via shared memory.
- Data can be partitioned among processors dynamically, depending on the load situation.

The main disadvantage of MPP systems for parallel databases is the necessity of static partitioning. This means it is difficult to vary the number of partitions, it is hard to size the partitions such that they are approximately equal for a wide range of operations, and there is basically no way to exercise control over difficult value distributions. As an example, consider hash joins. Re-partitioning the tuples based on the hash function in an MPP-system means that each tuple is sent via messages to the processor that represent the respective hash class.

Dynamic partitions, which can be created in SMP-architectures, allow to assign data to processors on demand, one can vary the number of logical processors (tasks) at run-time, and heavily skewed value distributions have the only effect of not creating the same load for each task. Hash joins in such a system are performed such that the shared memory is partitioned into n hash classes, and tuples are simply put into the partition corresponding to their hash class.

So SMP systems are more flexible in terms of load balancing, the number of options for parallel algorithms is higher because of the possibility of data sharing, and the need for reorganization is less than with MPP systems. Manageability is also easier for SMP systems. The big advantage of MPP systems, on the other hand, is their scalability over wide ranges of processor numbers.

6. SUMMARY AND CONCLUSION

In this paper we have discussed a new mechanism of parallel query execution in which a single operator is used for parallelizing a query execution plan. As compared to bracket model, all issues of control are localized in one operator (parallelizing operator) in this model. Also we can execute a complex query in a single or with a number of processes by using one or more parallelism operators in the query evaluation plan. Other operators did not require any modifications when we introduce a parallelizing operator a sequential query execution plan. For extensible database systems, the parallelizing operator presents an effective means for exploiting parallelism and parallel architectures, thus permitting database system designers to combine extensibility and high functionality with parallelism and high performance. We have noted that the objectives of parallel query processing cannot be achieved merely by the use of parallel machines, it is necessary to exploit the inherent parallelism available within queries to efficient utilize the resources, in particular, processors to achieve the projected speedup and throughput of database systems.

REFERENCES

[1] C. K. Baru, and O. Frieder, "Database operations in a cube-connected multicomputer system", *IEEE Trans. Comput*, vol. 38, No., 1989, pp. 920-927.

[2] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez, "Prototyping Bubba, a highly parallel database system", *IEEE Trans. Knowledge Data Engg.*, Vol. 2, No. 1,1990, pp. 4-24.

[3] W. Davison, "Parallel index building in Informix OnLine 6.0*", Proceeding of 1992 ACM Sigmod Conference, San Deigo*, 1992, pp. 103-103.

[4] D. J. Dewitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, and M. Muralikrishna, 1986, "GAMMA - A high performance dataflow database machine",. *Proceedings of the International 1986 Conference on Very Large Data Bases. VLDB Endowment*, 1986, pp. 228-237 .

[5] D. J. Dewitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. I. Hsiao, and R. Rasmussen, "The Gamma Database Machine Project", *IEEE Trans. Knowledge Data Engg.*, Vol. 2, No. 1, 1990, pp. 44-62.

[6] D. J. Dewitt, "The Wisconsin benchmark: Past, present, and future", *The Benchmark Handbook* , 1991, pp. 119-165.

[7] D. J. Dewitt, J. Naughton, and D. Schneider, "Parallel sorting on a shared-nothing architecture using probabilistic splitting", *Proceedings of the International 1991 Conference on Parallel and Distributed Information Systems, Miami Beach, Fla*, 1991, pp. 280-291.

[8] M. Gondran, and M. Minoux, "Graphs and Algorithms", *Interscience Series in Discrete Mathematics. Wiley,1984.*

[9] K.A. Hua, and C. Lee, "Handling data skew in multicomputer database computers using partition tuning", *Proceedings of 1991 International Conference on Very Large Data Bases. VLDB Endowment, 525, Barcelona, Spain,*1991, pp. 525-535.

[10] T. Keller, G. Graefe, and D. Maier, "Efficient assembly of complex objects", *ACM SIGMOD Record*, Vol. 20, No. 2, 1991, pp. 148-157.

[11] M. Kitwregawa, and Y. Ogawa, "Bucket spreading parallel hash: A new, robust, parallel hash join method for skew in the super database computer (SDC)", *Proceedings of 1990 International Conference on Very Large Data Bases. VLDB Endowment, 210, Brisbane, Australia*, 1990, pp. 210-221.

[12] M. S Lakshmi and P. S. Yu, "Effect of skew on join performance in parallel architectures", *Proceedings of 2000 International Symposium on Databases in Parallel and Distributed Systems, Austin, Texas*, 2000, pp. 107-120.

[13] M. S. Lakshmi and P. S. Yu, "Effectiveness of parallel joins", *IEEE Trans. Knowledge Data Engg*., Vol. 2, No. 4, 1990, pp. 410-424.

[14] E. Omiecinski, "Performance analysis of a load balancing relational hash-join algorithm for a shared-memory multiprocessor", *Proceedings of 1990 International Conference on Very Large Data Bases, VLDB Endowment, 375, Barcelona, Spain*, 1991, pp. 375-385.

[15] J. E. Richardson and M. J. Carey, "Programming Constructs for Database System Implementation in Exodus", *Proceedings of 1997 ACM SIGMOD Conference, CA, San Francisco*, 1987, pp. 208-219.

[16] P. G. Selinger, M. M. Astrahan, D. D. Chamberlain, R. A. Lorie, and G. Prwe, "Access path selection in a relational database management system", *Proceedings of 1979 ACM SIGMOD Conference. ACM, 23, New York*, 1979, pp: 23-34.

[17]] S. Seshadri and J. F. Naughton, "Sampling issues in parallel database systems", *Proceedings of 1992 International Conference on Extending Database Technology, Vienna, Austria*, 1992, pp. 328-343.

[18] C. B. Walton, "Investigating skew and scalability in parallel joins", *Computer Science Tech. Rep. 89-39, Univ. of Texas, Austin*, 1989.

[19] C. B. Walton, A. G. Dale and R. M. Jenevein, "A taxonomy and performance model of data

skew effects in parallel joins", *Proceedings of 1991 International Conference on Very Large Data Bases. VLDB Endowment, 537, Barcelona, Spain,* 1991, pp. 537-548.

[20] J. L. Wolf, D. M. Dias and P S. Yu, "An effective algorithm for parallelizing sort merge in the presence of data skew", *Proceedings of 1990 International Symposium on Database in Parallel and Distributed Systems, Dublin, Ireland,* 1990, pp. 103-115.

[21] J. L. Wolf, D. M. Dias, P. S. Yu and J. Turek, " An effective algorithm for parallelizing hash joins in the presence of data skew", *Proceedings of 1991 IEEE Conference on Data Engineering, New York,* 1991, pp. 200-209.