Parallel Database Systems:
The Future of Database Processing or a Passing Fad?

David J. DeWitt[1]
Computer Sciences Department
University of Wisconsin

Jim Gray
Tandem Computers Inc.

Abstract: The concept of parallel database machines consisting of exotic hardware has been replaced by a fairly conventional shared-nothing hardware base along with a highly parallel dataflow software architecture. Such a design provides speedup and scaleup in processing relational database queries. This paper reviews the techniques used by such systems, and surveys current commercial and research systems.

1 Introduction

The 1983 paper titled *Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines* [BORA83] painted a gloomy picture of the future of highly parallel database machines for two reasons. First, it observed that disk throughput was predicted to double while processor speeds were predicted to increase by a factor of ten to one hundred times. Consequently, it predicted that systems with multiple processors would soon be bottlenecked by their I/O systems unless a solution to the I/O bottleneck were found. Second, it predicted that none of the then trendy mass storage alternatives such as bubble or CCD memories were likely to replace standard moving head disks. At that time, the database machine field was much like the fashion industry — one fad after another. In particular, every time a new memory technology appeared it was soon followed by many paper designs, which all too frequently claimed to solve all the world's database problems.

While these predictions were fairly accurate about the future of hardware, the predictions were certainly wrong about the overall future of parallel database systems as evidenced by the success of Teradata and Tandem in developing and marketing highly parallel database machines. What happened? Certainly it has not been due to a fundamental change in the relative performance of CPUs and disks. If anything, the spread in their relative performance has grown even larger since 1983. In the interim, disk drives have improved their response time and throughput by about a factor of two, while in the same period single chip microprocessors have gone from about 1 mips in 1983 to well over twenty mips in 1990. Fortunately (or perhaps unfortunately depending on your perspective), many of these additional cycles have been consumed by corresponding increases in the complexity and sophistication of the database software itself.

Why then have parallel database systems become more than a research curiosity? One possible explanation is the widespread adoption of the relational data model. In 1983, relational database systems were just beginning to appear in the commercial market place. Today, they dominate it. In addition, mainframe vendors have found it increasingly difficult to build machines that are powerful enough to meet the CPU and I/O demands of relational DBMS serving large numbers of simultaneous users or searching terabyte databases. During this same time frame, multiprocessors based on increasingly fast and inexpensive microprocessors became widely available from a variety of vendors including Encore, Sequent, Tandem, Intel, Teradata, and NCUBE. These machines provide not only more total power than their mainframe counterparts, but also provide a lower price/MIPS. Also, their modular architectures enable customers to grow their configurations incrementally, adding mips, memory, and disks to a system either to speed up the processing of a given job, or to scale up the system to process a larger job in the same time.

The real answer is that special-purpose database *machines* have indeed failed. But, parallel database systems have been a big success. They have emerged as the major consumers of highly parallel architectures, and are in an excellent position to exploit massive numbers of fast-cheap commodity disks, processors, and memories. In fact, the successful parallel database systems are built from conventional processors, memories, and disks. Relational queries are ideally suited to parallel execution, consisting of uniform operations applied to uniform streams of data.

---

[1] This research was partially supported by the Defense Advanced Research Projects Agency under contract N00039-86-C-0578, by the National Science Foundation under grant DCR-8512862, and by research grants from Intel Scientific Computers, Tandem Computers, and Digital Equipment Corporation.

They require a message-based client-server operating system and also a high-speed network in order to interconnect the parallel processors. Such facilities may have seemed exotic a decade ago, but now they are clearly the direction that computer architecture is heading. The client-server paradigm based on high-speed LANs is the basis for most PC and workstation workgroup software, as well as the basis for distributed database technology.

A consensus on parallel and distributed database system architecture has emerged. This architecture is based on a *shared-nothing* hardware design [STON86] in which the only way processors communicate with one another is by sending messages via an interconnection network. In such systems, tuples of each relation in the database are *declustered* [LIVN87] across disk drives attached directly to each processor. Declustering allows multiple processors to scan large relations in parallel without any exotic I/O devices. This design is used by Tandem [TAND87, TAND88], Teradata [TERA83, TERA85], Gamma [DEWI86, DEWI90], Bubba [BORA90], Arbre [LORI89], and several other systems currently under development. Such architectures were pioneered by Teradata in the late seventies as well as by the Muffin [STON79], Xtree [GOOD81], and MDBS [BOYN83] projects.

2. Basic Techniques for Parallel Database Machine Implementation

Technology is driving parallel database machines. The ideal machine would be a single infinitely fast processor with an infinite memory with infinite bandwidth — and it would be infinitely cheap (free). Given such a machine, the need for parallelism would be non-existent. Unfortunately, technology is not delivering such machines — but it is coming close. Technology is promising to deliver fast one-chip processors, fast and large disks, and large ram memories. And each of these devices will be very cheap by today's standards, costing only hundreds of dollars each. *So the challenge is to build an infinitely fast processor out of infinitely many processors of finite speed, and to build an infinitely large memory with infinite memory bandwidth from infinitely many storage units of finite speed.* This sounds trivial mathematically; but in practice, when a new processor is added to most computer designs, it slows every other computer down just a little bit. If this slowdown is 1%, then when adding a processor to a 100 processor complex, the added power is canceled by the slowdown of the other processors. Such a system has a maximum speedup of 25 (at 50 processors) and the 100 processor complex has the effective power of a single processor system.

The ideal parallel system demonstrates two key properties: (1) *linear speedup:* Twice as much hardware can perform the task in half the elapsed time, and (2) *linear scaleup*: Twice as much hardware can perform twice as large a task in the same elapsed time.

The barriers to these goals are the following: (1) *startup* — The time needed to start a parallel operation. If thousands of processes must be started, this can easily dominate the actual computation time. (2) *interference* — The slowdown each new process imposes on all others. (3) *skew* — The service time of a job is often the service time of the slowest step of the job. As the number of parallel steps increases the average size of each step decreases, but the variance can well exceed the mean. When the variance dominates the mean, further parallelism will not help.

2.1 Shared-Nothing vs. Shared-Memory

There are a number of reasons for the success of shared-nothing systems. The main advantage of shared-nothing multiprocessors is that they can be scaled up to hundreds and probably thousands of processors which do not interfere with one another. Teradata and Tandem, for example, have shipped systems with more than 200 processors, and Intel is implementing a 2000 node Hypercube. On the other hand, the largest shared memory multiprocessors currently available are limited to about 32 processors.

Interference is a major problem for shared-memory multiprocessors for database applications. As processors become faster, each processor is given a large private cache in order to avoid interference on shared memory resources. Recent results measuring a multiprocessor running an online transaction workload show that the cache loading/flushing overhead of transaction processing applications considerably degrades processor performance [THAK90]. We do not believe high-performance shared-memory machines will scale beyond tens of processors when running database applications[2]. When high degrees of parallelism are introduced, shared resources become a bottleneck. Consequently the software architecture must use fine granularity concurrency control, and must even partition resources to avoid process and processor interference. This observation is true for both shared-nothing and

[2] SIMD machines such as ILLIAC IV and its derivatives like the Connection Machine are ignored here because to date no software designer has devised a programming technique for them. SIMD machines seem to have application in simulation, pattern matching, and mathematical search, but they do not seem to be appropriate for the i/o intensive and dataflow paradigm of database systems.

shared memory designs. This partitioning on a shared-memory system creates many of the problems faced by a shared nothing machine.

As demonstrated by Teradata in many competitive benchmarks, and by [DEWI90] and [TAND88, ENGL89], it is possible to obtain near-linear speedups on complex relational queries as well as for online-transaction processing using a shared-nothing architecture. Given such results, there is little justification for bothering with the hardware and software complexity associated with a shared-memory design.

2.2 Software

Two software mechanisms have been used in the construction of most parallel database systems: *declustering* [LIVN87] for mapping tuples to disks and an *intra-operator parallelizer* to distribute the tuples of intermediate relations among the processors being used to execute a relational operator.

Declustering

Declustering a relation involves distributing the tuples of a relation among two or more disk drives according to some distribution criteria such as applying a hash function to the key attribute of each tuple. Declustering has its origins in the concept of *horizontal partitioning* initially developed as a distribution mechanism for distributed DBMS [RIES78]. One of the key reasons for using declustering in a parallel database systems is to enable the DBMS software to exploit the I/O bandwidth reading and writing multiple disks in parallel. By declustering the tuples of a relation the task of parallelizing a scan operator becomes trivial. All that is required is to start a copy of the operator on each processor or disk containing relevant tuples, and to merge their outputs at the destination. For operations requiring a sequential scan of an entire relation, this approach can provide the same I/O bandwidth as a RAID-style system [SALE84], [PATT88] **without** needing any specialized hardware.

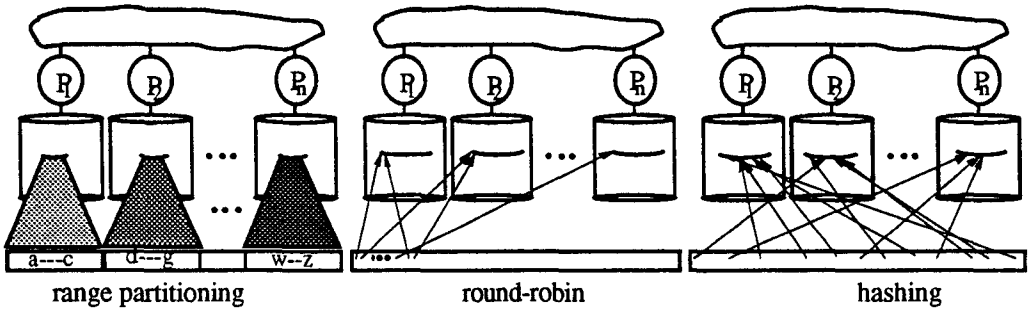

Figure 1: The three basic declustering schemes: range declustering maps contiguous fragments of a table to various disks. Round-Robin declustering maps the i'th record to disk i **mod** n. Hashed declustering, maps each record to a disk location based on some hash function. Each of these schemes spreads data among a collection of disks, allowing parallel disk access and parallel processing.

While tuples can simply be declustered in a *round-robin* fashion, more interesting alternatives exist (see Figure 1). One is to apply a *hashing* function to the key attribute of each tuple to distribute the tuples among the disks. This distribution mechanism allows exact match selection operations on the partitioning attribute to be directed to a single disk, avoiding the overhead of starting such queries on multiple disks. On the other hand, range queries on the partitioning attribute, must be sent to all disks over which a relation has been declustered. A hash declustering mechanism is provided by Arbre, Bubba, Gamma, and Teradata.

An alternative declustering strategy is to associate a distinct range of partitioning attribute values with each disk by, for example, dividing the range of possible values into N units, one for each of the N processors in the system *range partitioning*. The advantage of range declustering is that it can isolate the execution of both range and exact match-selection operations to the minimal number of processors. Another advantage of range partitioning is that it can be used to deal with non-uniformly distributed partitioning attribute values. A range-partitioning mechanism is provided by Arbre, Bubba, Gamma, and Tandem.

While declustering is a simple concept that is easy to implement, it raises a number of new physical database design issues. In addition to selecting a declustering strategy for each relation, the number of disks over which a relation should be declustered must be decided. While Gamma declusters all relations across all disk drives (primarily to simplify the implementation), the Bubba, Tandem, and Teradata systems allow a subset of the disks to

be used. In general, increasing the degree of declustering reduces the response time for an individual query and (generally) increases the overall throughput of the system. For sequential scan queries, the response time decreases because more processors and disks are used to execute the query. For indexed selections on the partitioning attribute, the response time improves because fewer tuples are stored at each node and hence the size of the index that must be searched decreases. However, there is a point beyond which further declustering actually increases the response time of a query. This point occurs when the cost of starting a query on a node becomes a significant fraction of the actual execution time [COPE88, DEWI88, GHAN90a]. In general, full declustering is not always a good idea, especially in very large configurations. Bubba [COPE88] refines the concept of range-partitioning by considering the **heat**[3] of its tuples when declustering a relation; the goal is to balance the frequency with which each disk is accessed rather than the actual number of tuples on each disk. In [COPE88] the effect of the degree of declustering on the multiuser throughput of Bubba is studied. In [GHAN90a] the impact of the alternative partitioning strategies on the multiuser throughput of Gamma is evaluated.

Intra-operator Parallelization

The parallelization of selection operators should now be obvious. The approach used in DIRECT [DEWI79] to execute such dataflow graphs in parallel was to design and implement parallel algorithms for each operator. Fortunately, a much simpler approach is possible[4] [DEWI86]. The basic idea is to parallelize the data instead of parallelizing the operators (programs), enabling the use of existing sequential routines to execute the relational operators. Consider first the simple scan of a table called A that has been declustered across three disks into partitions A0, A1, and A2. This scan can be implemented as three scan operators which send their output to a common merge operator which produces a single output data stream to the application or to the next relational operator. So the first basic operator is a *merge* operator that can combine several parallel data streams into a single sequential stream. The parallel query executor creates three processes and directs them to take their inputs from some sequential stream and direct their outputs to the merge node. Each of the scans can run on an independent processor and disk.

The merge operator tends to focus data on one spot. Clearly, if a multi-stage operation is to be done in parallel, a single data stream must be split into several independent streams. To do this, a structure known as a *split table* is used to distribute the stream of tuples produced by a relational operator. A split table defines a mapping of one or more attribute values of the output tuples produced by a process executing a relational operator to a set of destination processes.

As an example, consider the two split tables shown in Figure 2 in conjunction with the query shown in Figure 3. Assume that three processes are used to execute the join operator, and that five other processes execute the two scan operators — three scanning partitions of table A while two scan partitions of table B. Each of the three table A scan nodes will have the same split table, sending all tuples between "A-H" to port 1 of join process 0, all between "I-Q" to port 1 of join process 1, and all between "R-Z" to port 1 of join process 2. Similarly the two table B scan nodes have the same split table except that their outputs are merged by port 1 (not port 0) of each join process. Each join process sees a sequential input stream of A tuples from the port 0 merge (the left scan nodes) and another sequential stream of B tuples from the port 1 merge (the right scan nodes).

Table A Scan Split Table		Table B Scan Split Table	
Predicate	Destination Process	Predicate	Destination Process
"A-H"	(cpu #5, Process #3, Port #0)	"A-H"	(cpu #5, Process #3, Port #1)
"I-Q"	(cpu #7, Process #8, Port #0)	"I-Q"	(cpu #7, Process #8, Port #1)
"R-Z"	(cpu #2, Process #2, Port #0)	"R-Z"	(cpu #2, Process #2, Port #1)

Figure 2. Sample Split Tables which map tuples to different output streams (ports of other processes) depending on the range value of some attribute of the input tuple.The split table on the left is for the Table A scan in figure 7, while the table on the right is for the table B scan.

[3]The **heat** of an object is the access frequency of the object over some period of time" [COPE88]

[4]Teradata may use a similar mechanism but, if so, it has not been described publicly.

The split tables in Figure 2 are just an example. Other split tables might duplicate the input stream, or partition it round-robin, or partition it by hash. The partitioning function is arbitrary. In fact in Volcano, it is an arbitrary program [GRAE90]. For each process executing the scan, the split table applies the predicates to the join attribute of each output tuple. If the predicate is satisfied, the tuple is sent to the corresponding destination (this is the logical model, the actual implementation is generally optimized by table lookups). Each join process has two ports, each of which merges the outputs from the various scan split tables.

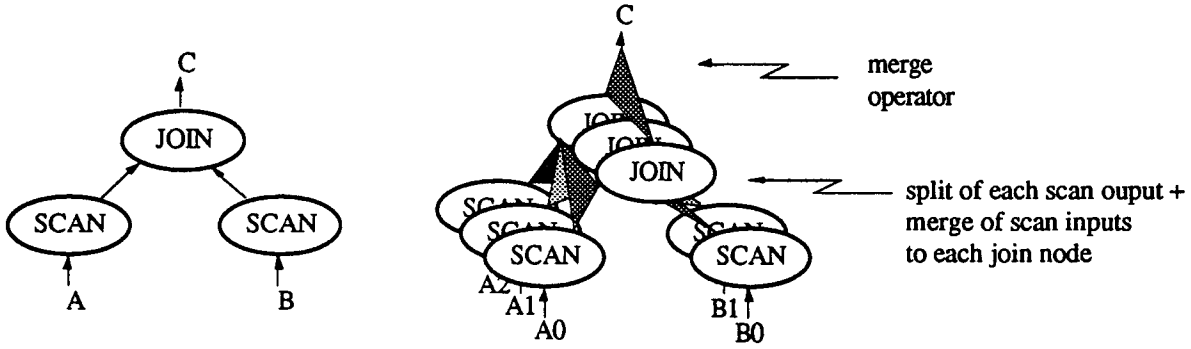

Figure 3: A simple relational dataflow graph showing two relational scans (project and select) consuming two input tables, A and B and feeding their outputs to a join operator which in turn produces a data stream C.

Both Volcano and Tandem have taken Gamma's notion of a split table one step further and made it a separator operator that can be inserted anywhere in a query tree [GRAE90]. This approach has a number of advantages including the automatic parallelism of any new operator added to the system plus support for a variety of different kinds of parallelism. In addition, in Volcano this encapsulation made it possible to eliminate the need for a separate scheduler process such as that used by Gamma to coordinate the processes executing the operators of the query. Rather the flow control and buffering are built into the merge and join operators.

For simplicity, these examples have been stated in terms of an operator per process. But it is entirely possible to place several operators within a process to get coarser grained parallelism. The fundamental idea though is to build a self-pacing dataflow graph and distribute it in a shared-nothing machine in a way that minimizes interference.

3. Future Directions and Research Problems

3.1.Mixing Batch and OLTP Queries

Section 2 concentrated on the basic techniques used for processing complex relational queries in a parallel database system. Tandem and Teradata have demonstrated that the same architecture can be used successfully to process transaction-processing workloads [TAND88] as well as ad-hoc queries [ENGB89]. Running a mix of both types of queries concurrently, however, presents a number of unresolved problems. The problem is that ad-hoc relational queries tend to acquire a large number of locks (at least from a logical viewpoint) and tend to hold them for a relatively long period of time (at least from the viewpoint of a competing debit-credit style query). The solution currently offered is "Browse mode" locking for ad-hoc queries that do not update the database. While such a "dirty-read" solution is acceptable for some applications, it is not universally acceptable. The solution advocated by XPRS [STON88] is to use a versioning mechanism to enable readers to read a consistent version of the database without setting any locks. While this approach is intuitively appealing, the associated performance implications need further exploration. Other, perhaps better, solutions for this problem may also exist.

A related issue is priority scheduling in a shared-nothing architecture. Even in centralized systems, batch jobs have a tendency to monopolize the processor, flood the memory cache, and make large demands on the I/O subsystem. It is up to the underlying operating system to quantize and limit the resources used by such batch jobs in order to insure short response times and low variance in response times for short transactions. A particularly difficult problem is the *priority inversion problem*, in which a low-priority client makes a request to a high priority server. The server must run at high priority because it is managing critical resources. Given this, the work of the low priority client is effectively promoted to high priority when the low priority request is serviced by the high-priority server. There have been several ad-hoc attempts at solving this problem, but considerably more work is needed.

3.2. Parallel Query Optimization

Currently, the query optimizers for most parallel database systems/machines do not consider alternative query tree formats when optimizing a relational query. Typically only alternative left-deep query trees are considered and not right deep or bushy trees. [GRAE89] proposes to dynamically select from among a number of alternative plans at run time depending on, for example, the amount of physical memory actually available and the cardinalities of the intermediate results. While cost models for relational queries running on a single processor are now well-understood [SELI79, JARK84, MACK86] they still depend on cost estimates that often are guesses. The situation with parallel join algorithms running in a mixed batch and online environment is even more complex. Only recently have we begun to understand the relative performance of the various parallel join methods [SCHN89] and alternative query tree organizations in a parallel database machine environment [SCHN90]. The XPRS optimizer considers, for example, only the use of merge-sort join methods. To date, no query optimizers consider the full variety of parallel algorithms for each operator and the full variety of alternative query tree organizations. While the necessary query optimizer technology exists, accurate cost models have not been developed, let alone validated. More work is needed in this area.

3.3. Parallelization of the Application Program

While machines like Teradata and Gamma separate the application program running on a host processor from the database software running on the parallel processor, both the Tandem and Bubba systems use the same processors for both application programs and the parallel database software. The latter arrangement has the disadvantage of requiring a complete, full-function operating system on the parallel processor, but it avoids any potential load imbalance between the two systems and allows parallel applications. Missing, however, are tools that would allow the application programs to take advantage of the inherent underlying parallelism of these integrated parallel systems. While automatic parallelization of applications programs written in Cobol may not be feasible, library packages to facilitate explicitly parallel application programs are needed. Support for the SQL3 NOWAIT option in which the application can launch several SQL statements at once would be an advance. Ideally the SPLIT and MERGE operators could be packaged so that applications could benefit from them.

3.4. Physical Database Design

As discussed in Section 3, Gamma currently provides four alternative declustering strategies in addition to the normal collection of access methods. While this is a richer set than what is currently available commercially, the results in [GHAN90a, GHAN90b] demonstrate that there is no one "best" declustering strategy. In addition, for a given database and workload there are a variety of viable alternative indexing and declustering combinations. Database design tools are needed to help the database administrator select the correct combination. Such a tool might accept as input a description of the queries comprising the workload (including their frequency of execution), statistical information about the relations in the database, and a description of the target environment. The resulting output would include a specification of which declustering strategy should be used for each relation (including which nodes the relation should be declustered over) plus a specification of the indexes to be created on each relation. Such a tool would undoubtedly need to be integrated with the query optimizer as query optimization must incorporate information on declustering and indexes which, in turn, impact what plan is chosen for a particular query.

Another area where additional research is needed is multidimensional declustering algorithms. All current algorithms decluster the tuples in a relation using the values of a single attribute. While this arrangement allows selections against the partitioning attribute to be localized to a limited number of nodes, selections on any other attribute must be sent to all the nodes over which the relation is declustered for processing. While this is acceptable in a small configuration, it is not in a system with thousands of processors. Recently we have begun to study the applicability of grid file techniques to the problem of multidimensional declustering. While the initial results look promising, the associated database design problem of computing the proper size of each bucket and the distribution of buckets among processors appears daunting.

3.5. On-line Reorganization and Utilities

Loading, reorganizing, or dumping a terabyte database at a megabyte a second takes over 12 days and nights. Clearly parallelism is needed if utilities are to complete within a few hours or days. Even then, it will be essential that the data be available while the utilities are operating. In the SQL world, typical utilities create indexes, add or drop columns, add constraints, and physically reorganize the data, changing its clustering.

One unexplored (and difficult) problem is how to process database utility commands while the system

remains operational and the data remains available for concurrent reads and writes by others. A technique for reorganizing indexes was proposed in [STON88]. The fundamental properties of such algorithms is that they must be *online* (operate without making data unavailable), *incremental* (operate on parts of a large database), *parallel* (exploit parallel processors), and *recoverable* (allow the operation to be canceled and return to the old state).

3.6. Processing Highly Skewed Data

Another interesting research area is algorithms to handle relations with highly skewed attribute value distributions. Both range declustering and hybrid-range declustering with DBA-supplied key ranges help alleviate the problem of declustering a relation whose partitioning attribute value is skewed (especially if the heat of the tuples in the relation is considered [COPE88]). For example, there is really no reason each node must have a unique range of partitioning attribute values when declustering a relation using range partitioning. Problems can still occur, however, when the data is redistributed as part of processing a complex operator such as a join. One possible solution is to use a range-split table with non-unique entries for redistributing the tuples of two relations to be joined. In the case of the inner relation, if a tuple's joining attribute value falls into one of the non-unique ranges, one entry is picked at random from the set of possible alternatives. Tuples from the outer relation need, however, to be sent to all the possible alternatives. Other approaches to solving this problem have been proposed in [KITS90, WOLF90] and certainly other solutions remain to be explored and evaluated.

3.7. Non-relational Parallel Database Machines

While open research issues remain in the area of parallel database machines for relational database systems, building a highly parallel database machine for an object-oriented database system presents a number of new challenges. One of the first issues to resolve is how declustering should be handled. For example, should one decluster all sets (such as set-valued attributes of a complex object) or just top-level sets? Another question is how inter-object references should be handled. In a relational database machine, such references are handled by doing a join between the two relations of interest, but in an object-oriented DBMS references are generally handled via object identifiers. In particular, a tension exists between declustering a set in order to parallelize scan operations on that set and clustering an object and the objects it references in order to reduce the number of disk accesses necessary to access the components of a complex object. Since clustering in an object-oriented database system remains an open research issue, mixing in declustering makes the problem even more challenging.

Another open area is parallel query processing in an OODBMS. Various OODBMS provide a relational-like query language based on an extension to relational algebra. While it is possible to parallelize these operators, how should class-specific methods be handled? If the method operates on a single object it is certainly not worthwhile to parallelize it. However, if the method operates on a set of values or objects that are declustered, then it almost must be parallelized if one is going to avoid moving all the data referenced to a single processor for execution. Since it is, at this point in time, impossible to parallelize arbitrary method code, one possible solution might be to insist that, if a method is to be parallelized, it be constructed using the primitives from the underlying algebra, perhaps embedded in a normal programming language.

4. Summary and Conclusions

Like most applications, database systems will run on cheap, fast hardware. Today that means commodity processors, memories, and disks. Consequently, the hardware concept of a *database machine* built of exotic hardware is inappropriate for current technology. On the other hand, the availability of fast microprocessors, and small inexpensive disks packaged as standard inexpensive but fast computers is an ideal platform for *parallel database systems*. A shared-nothing architecture is relatively straightforward to implement and, more importantly, has demonstrated both speedup and scaleup to hundreds of processors. Furthermore, shared-nothing architectures actually simplify the software implementation. If the software techniques of data declustering and intra-operator parallelizers are employed, the task of parallelizing an existing database management system becomes a relatively straightforward task. Finally, there are certain applications (e.g. data mining in terabyte databases) that require the computational and I/O resources only available from a parallel architecture.

While the successes of both the commercial products and prototypes demonstrate the viability of highly parallel database machines, a number of research issues remain unsolved including techniques for mixing ad-hoc queries and online transaction processing without seriously limiting transaction throughput, improved optimizers for parallel queries, tools for physical database design and on-line database reorganization, and algorithms for handling relations with highly skewed data distributions. Some application domains are not well-supported by the relational data model, and it appears that a new class of parallel database systems based on an object-oriented data model are

needed. Such systems pose a host of interesting research problems that require further examination.

References

[ALEX88] Alexander, W., et. al., "Process and Dataflow Control in Distributed Data-Intensive Systems," Proc. ACM SIGMOD Conf., Chicago, IL, June 1988. October, 1983.

[BLAS79] Blasgen, M. W., Gray, J., Mitoma, M., and T. Price, "The Convoy Phenomenon," Operating System Review, Vol. 13, No. 2, April, 1979.

[BOYN83] Boyne, R.D., D.K. Hsiao, D.S. Kerr, and A. Orooji, A Message-Oriented Implementation of a Multi-Backend Database System (MDBS), Proceedings of the 1983 Workshop on Database Machines, edited by H.-O. Leilich and M. Missikoff, Springer-Verlag, 1983.

[BORA83] Boral, H. and D. DeWitt, "Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines," Proceedings of the 1983 Workshop on Database Machines, edited by H.-O. Leilich and M. Missikoff, Springer-Verlag, 1983.

[BORA90] Boral, H. et. al., "Prototyping Bubba: A Highly Parallel Database System," IEEE Knowledge and Data Engineering, Vol. 2, No. 1, March, 1990.

[COPE88]Copeland, G., Alexander, W., Boughter, E., and T. Keller, "Data Placement in Bubba," Proceedings of the ACM-SIGMOD International Conference on Management of Data, Chicago, May 1988.

[DEWI79] DeWitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," IEEE Transactions on Computers, June, 1979.

[DEWI86] DeWitt, D., et. al., "GAMMA - A High Performance Dataflow Database Machine," Proceedings of the 1986 VLDB Conference, Japan, August 1986.

[DEWI88] DeWitt, D., Ghandeharizadeh, S., and D. Schneider, "A Performance Analysis of the Gamma Database Machine," Proceedings of the ACM-SIGMOD International Conference on Management of Data, Chicago, May 1988.

[DEWI90] DeWitt, D., et. al., "The Gamma Database Machine Project," IEEE Knowledge and Data Engineering, Vol. 2, No. 1, March, 1990.

[ENGL89] Englert, S, J. Gray, T. Kocher, and P. Shah, "A Benchmark of NonStop SQL Release 2 Demonstrating Near-Linear Speedup and Scaleup on Large Databases," Tandem Computers, Technical Report 89.4, Tandem Part No. 27469, May 1989.

[GHAN90a] Ghandeharizadeh, S., and D.J. DeWitt, "Performance Analysis of Alternative Declustering Strategies", Proceedings of the 6th International Conference on Data Engineering, Feb. 1990.

[GHAN90b] Ghandeharizadeh, S. and D. J. DeWitt, "Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines" Proceedings of the Sixteenth International Conference on Very Large Data Bases", Melbourne, Australia, August, 1990.

[GOOD81] Goodman, J. R., "An Investigation of Multiprocessor Structures and Algorithms for Database Management", University of California at Berkeley, Technical Report UCB/ERL, M81/33, May, 1981.

[GRAE89] Graefe, G., and K. Ward, "Dynamic Query Evaluation Plans", Proceedings of the 1989 SIGMOD Conference, Portland, OR, June 1989.

[GRAE90] Graefe, G., "Encapsulation of Parallelism in the Volcano Query Processing System," Proceedings of the 1990 ACM-SIGMOD International Conference on Management of Data, May 1990.

[JARK84] Jarke, M. and J. Koch, "Query Optimization in Database System," ACM Computing Surveys, Vol. 16, No. 2, June, 1984.

[KIM86] Kim, M., "Synchronized Disk Interleaving," IEEE Transactions on Computers, Vol. C-35, No. 11, November 1986.

[KITS90] Kitsuregawa, M., and Y. Ogawa, "A New Parallel Hash Join Method with Robustness for Data Skew in Super Database Computer (SDC)", Proceedings of the Sixteenth International Conference on Very Large Data Bases", Melbourne, Australia, August, 1990.

[LIVN87] Livny, M., S. Khoshafian, and H. Boral, "Multi-Disk Management Algorithms", Proceedings of the 1987 SIGMETRICS Conference, Banff, Alberta, Canada, May, 1987.

[LORI89] Lorie, R., J. Daudenarde, G. Hallmark, J. Stamos, and H. Young, "Adding Intra-Transaction Parallelism to an Existing DBMS: Early Experience", IEEE Data Engineering Newsletter, Vol. 12, No. 1, March 1989.

[MACK86] Mackert, L. F. and G. M. Lohman, "R* Optimizer Validation and Performance Evaluation for Local Queries," Proceedings of the 1986 SIGMOD Conference, Washington, D.C., May, 1986.

[PATT88] Patterson, D. A., G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," Proceedings of the ACM-SIGMOD International Conference on Management of Data, Chicago, May 1988.

[RIES78] Ries, D. and R. Epstein, "Evaluation of Distribution Criteria for Distributed Database Systems," UCB/ERL Technical Report M78/22, UC Berkeley, May, 1978.

[SALE84] Salem, K. and H. Garcia-Molina, "Disk Striping", Department of Computer Science Princeton University Technical Report EEDS-TR-332-84, Princeton N.J., Dec. 1984

[SCHN89] Schneider, D. and D. DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", Proceedings of the 1989 SIGMOD Conference, Portland, OR, June 1989.

[SCHN90] Schneider, D. and D. DeWitt, "Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines," Proceedings of the Sixteenth International Conference on Very Large Data Bases", Melbourne, Australia, August, 1990.

[SELI79] Selinger,P. G., et. al., "Access Path Selection in a Relational Database Management System," Proceedings of the 1979 SIGMOD Conference, Boston, MA., May 1979.

[STON79] Stonebraker, M., "Muffin: A Distributed Database Machine," ERL Technical Report UCB/ERL M79/28, University of California at Berkeley, May 1979.

[STON86] Stonebraker, M., "The Case for Shared Nothing," Database Engineering, Vol. 9, No. 1, 1986.

[STON88] Stonebraker, M., R. Katz, D. Patterson, and J. Ousterhout, "The Design of XPRS", Proceedings of the Fourteenth International Conference on Very Large Data Bases, Los Angeles, CA, August, 1988.

[TAND87] Tandem Database Group, "NonStop SQL, A Distributed, High-Performance, High-Reliability Implementation of SQL," Workshop on High Performance Transaction Systems, Asilomar, CA, September 1987.

[TAND88] Tandem Performance Group, "A Benchmark of Non-Stop SQL on the Debit Credit Transaction," Proceedings of the 1988 SIGMOD Conference, Chicago, IL, June 1988.

[TERA83] Teradata: DBC/1012 Data Base Computer Concepts & Facilities, Teradata Corp. Document No. C02-0001-00, 1983.

[TERA85] Teradata, "DBC/1012 Database Computer System Manual Release 2.0," Document No. C10-0001-02, Teradata Corp., NOV 1985.

[THAK90] Thakkar, S.S. and M. Sweiger, "Performance of an OLTP Application on Symmetry Multiprocessor System," Proceedings of the 17th Annual International Symposium on Computer Architecture, Seattle, WA., May, 1990.

[WOLF90] Wolf, J.L., Dias, D.M., and P.S. Yu, "An Effective Algorithm for Parallelizing Sort-Merge Joins in the Presence of Data Skew," 2nd International Symposium on Databases in Parallel and Distributed Systems, Dublin, Ireland, July, 1990.