**Parallel Database Systems:**

**The Future of High Performance Database Processing**

**Scope of the topic**

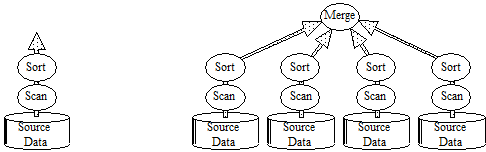
**Abstract**: Parallel database machine architectures have evolved from the use of exotic hardware to a software parallel dataflow architecture based on conventional shared-nothing hardware. These new designs provide impressive speedup and scaleup when processing relational database queries. This paper reviews the techniques used by such systems, and surveys current commercial and research systems.

**Literature Work**

Introduction parallel database system?

Why have parallel database systems become more than a research curiosity? One explanation is the widespread adoption of the relational data model. In 1983 relational database systems were just appearing in the marketplace; today they dominate it. Relational queries are ideally suited to parallel execution; they consist of uniform operations applied to uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graphs. By streaming the output of one operator into the input of another operator, the two operators can work in series giving *pipelined parallelism*. By partitioning the input data among multiple processors and memories, an operator can often be split into many independent operators each working on a part of the data. This partitioned data and execution gives *partitioned parallelism*

A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries. ... Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel.



**Figure 1. The dataflow approach to relational operators gives both pipelined and partitioned parallelism.**  Relational data operators take relations (uniform sets of records) as input and produce relations as outputs. This allows them to be composed into dataflow graphs that allow *pipeline parallelism* (left) in which the computation of one operator proceeds in parallel with another, and *partitioned parallelism* in which operators (sort and scan in the diagram at the right) are replicated for each data source, and the replicas execute in parallel.

**Paper : [ Parallel Database Systems: The Future of High Performance Database Processing, - , Proceedings of the Fourteenth International Conference on Very Large Data Bases]**

Note : Most of the content in this paper are selected form the papers published in “Proceedings of the Fourteenth International Conference on Very Large Data Bases “

# Basic Techniques for Parallel Database Machine Implementation

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 (see Figures 2 and 3).



**Figure 2. Speedup and Scaleup.** A speedup design performs a one-hour job four times faster when run on a four-times larger system. A scaleup design runs a ten-times bigger job is done in the same time by a ten-times bigger system.

More formally, given a fixed job run on a small system, and then run on a larger system, the *speedup* given by the larger system is measured as:

*Speedup =*

Speedup is said to be linear, if an *N*-times large or more expensive system yields a speedup of *N*.

Speedup holds the problem size constant, and grows the system. Scaleup measures the ability to grow both the system and the problem. *Scaleup* is defined as the ability of an *N*-times larger system to perform an *N*-times larger job in the same elapsed time as the original system. The scaleup metric is.

*Scaleup =*

If this scaleup equation evaluates to 1, then the scaleup is said to be linear[[1]](#footnote-1). There are two distinct kinds of scaleup, batch and transactional. If the job consists of performing many small independent requests submitted by many clients and operating on a shared database, then scaleup consists of *N*-times as many clients, submitting *N*-times as many requests against an *N*-times larger database. This is the scaleup typically found in transaction processing systems and timesharing systems. This form of scaleup is used by the Transaction Processing Performance Council to scale up their transaction processing benchmarks [GRAY91]. Consequently, it is called *transaction-scaleup*. Transaction scaleup is ideally suited to parallel systems since each transaction is typically a small independent job that can be run on a separate processor.

**Figure 2. Good and bad speedup curves.** The standard speedup curves. The left curve is the ideal. The middle graph shows no speedup as hardware is added. The right curve shows the three threats to parallelism. Initial startup costs may dominate at first. As the number of processes increase, interference can increase. Ultimately, the job is divided so finely, that the variance in service times (skew) causes a slowdown.

Section 2.3 describes several basic techniques widely used in the design of shared-nothing parallel database machines to overcome these barriers. These techniques often achieve linear speedup and scaleup on relational operators.

Hardware Architecture, the Trend to Shared-Nothing Machines

* The ideal database machine would have a single infinitely fast processor with an infinite memory with infinite bandwidth — and it would be infinitely cheap (free).
* Given such a machine, there would be no need for speedup, scaleup, or parallelism.
* . Technology is promising to deliver fast one-chip processors, fast high-capacity disks, and high-capacity electronic RAM memories.
* It also promises that each of these devices will be very inexpensive by today's standards, costing only hundreds of dollars each.
* Large-scale;-multiprocessor systems have long held the promise of substantially
* higher performance than traditional uniprocessor systems.
* However, due to a number of difficult problems, the potential of these machines
* has been difficult to realize.
* Given such results, database machine designers see little justification for the hardware and
* software complexity associated with shared-memory and shared-disk designs.
* Stonebraker suggested the following simple taxonomy for the spectrum of designs
* 1.shared-memory: All processors share direct access to a common global memory and to all disks. The IBM/370, and Digital VAX, and Sequent Symmetry multi-processors typify this design.
* 2.shared-disks: Each processor has a private memory but has direct access to all disks. The IBM Sysplex and original Digital VAXcluster typify this design.
* 3.shared-nothing: Each memory and disk is owned by some processor that acts as a server for that data. Mass storage in such an architecture is distributed among the processors by connecting one or more disks. The Teradata, Tandem, and nCUBE machines typify this design.
* • 1.shared-memory and 2. shared-disks
* 
* **Figure 4. The shared-memory and shared-disk designs.**  A shared-memory multi-processor connects all processors to a globally shared memory. Multi-processor IBM/370, VAX, and Sequent computers are typical examples of shared-memory designs. Shared-disk systems give each processor a private memory, but all the processors can directly address all the disks. Digital's VAXcluster and IBM's Sysplex typify this design.

3. shared-nothing

These shared-nothing architectures achieve near-linear speedups and scaleups on complex relational queries and on online-transaction processing workloads [DEWI90, TAND88, ENGL89]. Given such results, database machine designers see little justification for the hardware and software complexity associated with shared-memory and shared-disk designs.



**Figure 3.** **The basic shared-nothing design.**  Each processor has a private memory and one or more disks. Processors communicate via a high-speed interconnect network. Teradata, Tandem, nCUBE, and the newer VAXclusters typify this design.

A Parallel Dataflow Approach to SQL Software?

Relations are created, updated, and queried by writing SQL statements. These statements are syntactic sugar for a simple set of operators chosen from the relational algebra. *Select-project*, here called *scan*, is the simplest and most common operator – it produces a row-and-column subset of a relational table. A scan of relation *R* using predicate *P* and attribute list *L* produces a relational data stream as output. The scan reads each tuple, *t,* of *R* and applies the predicate *P* to it. If *P(t)* is true, the scan discards any attributes of *t* not in *L* and inserts the resulting tuple in the scan output stream. Expressed in SQL, a scan of a telephone book relation to find the phone numbers of all people named Smith would be written:

SELECT telephone\_number /\* the output attribute(s) \*/

FROM telephone\_book /\* the input relation \*/

WHERE last\_name = 'Smith'; /\* the predicate \*/

A scan's output stream can be sent to another relational operator, returned to an application, displayed on a terminal, or printed in a report. Therein lies the beauty and utility of the relational model. The uniformity of the data and operators allow them to be arbitrarily composed into dataflow graphs.

The output of a scan may be sent to a sort operator that will reorder the tuples based on an attribute sort criteria, optionally eliminating duplicates.

SQL defines several aggregate operators to summarize attributes into a single value.

for example

taking the sum, min, or max of an attribute, or counting the number of distinct values of the attribute

* The SQL data model was originally proposed to improve programmer productivity by offering a non-procedural database language
* . Data independence was an additional benefit; since the programs do not specify how the query is to be executed, SQL programs continue to operate as the logical and physical database schema evolves.

Data Partitioning

The simplest partitioning strategy distributes tuples among the fragments in a roundrobin fashion. This is the partitioned version of the classic entry-sequence file. Round robin partitioning is excellent if all applications want to access the relation by sequentially scanning all of it on each query. The problem with round-robin partitioning is that applications frequently want to associatively access tuples, meaning that the application wants to find all the tuples having a particular attribute value. The SQL query looking for the Smith's in the phone book is an example of an associative search.

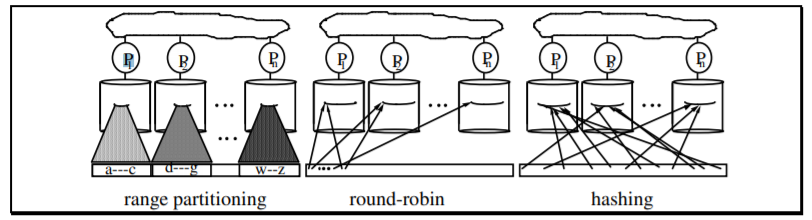


Figure 5: The three basic partitioning schemes. Range partitioning maps contiguous attribute ranges of a relation to various disks. Round-robin partitioning maps the i’th tuple to disk i mod n. Hashed partitioning, maps each tuple to a disk location based on a hash function. Each of these schemes spreads data among a collection of disks, allowing parallel disk access and parallel processing.

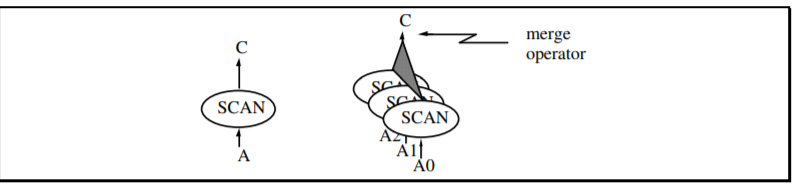


Figure 6: Partitioned data parallelism. A simple relational dataflow graph showing a relational scan (project and select) decomposed into three scans on three partitions of the input stream or relation. These three scans send their output to a merge node that produces a single data stream.

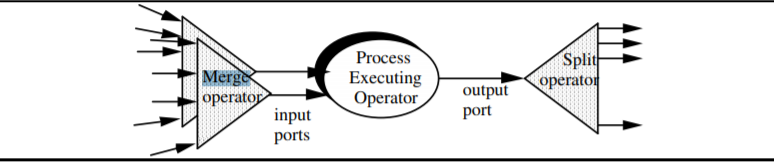


Figure 7: Merging the inputs and partitioning the output of an operator. A relational dataflow graph showing a relational operator’s inputs being merged to a sequential steam per port. The operator's output is being decomposed by a split operator into several independent streams. Each stream may be a duplicate or a partitioning of the operator output stream into many disjoint streams. With the split and merge operators, a web of simple sequential dataflow nodes can be connected to form a parallel execution plan.

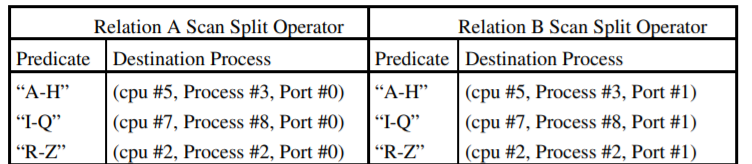


Figure 8. Sample split operators. Each split operator maps tuples to a set of output streams (ports of other processes) depending on the range value (predicate) of the input tuple. The split operator on the left is for the relation A scan in Figure 7, while the table on the right is for the relation B scan. The tables above partition the tuples among three data streams. If the predicates were TRUE for all the tuples, the split operator would replicate the tuples on all three output streams.

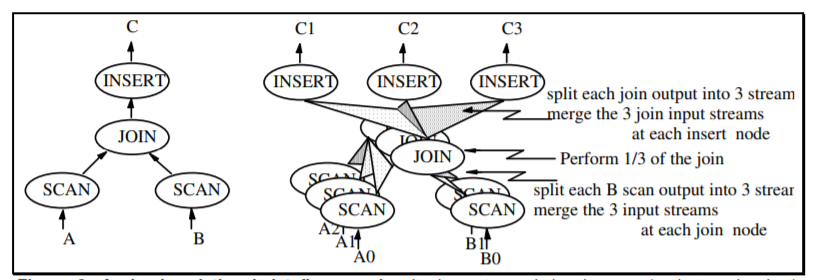


Figure 9: A simple relational dataflow graph. It shows two relational scans (project and select) consuming two input relations, A and B and feeding their outputs to a join operator that in turn produces a data stream C

**Conclusion :**

Like most applications, database systems want 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. While the successes of both commercial products and prototypes demonstrates the viability of highly parallel database machines, several open research issues remain unsolved including techniques for mixing ad-hoc queries and with online transaction processing without seriously limiting transaction throughput, improved optimizers for parallel queries, tools for physical database design, 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. It appears that a new class of database systems based on an object-oriented data model are needed. Such systems pose a host of interesting research problems that required further examination.

**Reference :**

Google Scholar

1. [↑](#footnote-ref-1)