

Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Semantic Data Control
- Distributed Query Processing
- Distributed Transaction Management
- Distributed Database Operating Systems
- Open Systems and Interoperability
- Parallel Database Systems
 - Parallel Architectures
 - Parallel Data Management Techniques
- Distributed Object Management
- Concluding Remarks

The Information System Problem

- Enterprises depend on timely available, up-to-date information
 - information volume growth : 30 % per year
 - transaction rate growth : 10× over next five years
- Transaction load is changing
 - simple OLTP-like transactions
 - complex transactions (e.g., generated by decision-support systems)
 - very complex transactions (e.g., generated by expert systems)
- The need: database servers that provide high-throughput and good response times for mixed workloads on very large on-line databases

The Database Problem

- large volume of data \Rightarrow use disk and large main memory
- I/O bottleneck (or memory access bottleneck)
 - \Rightarrow speed(disk) \ll speed(RAM) \ll speed(microprocessor)
- Predictions
 - \Rightarrow (micro-) processor speed growth : 50 % per year
 - \Rightarrow DRAM capacity growth : 4 \times every three years
 - \Rightarrow disk throughput : 2 \times in the last ten years
- Conclusion : the I/O bottleneck will worsen

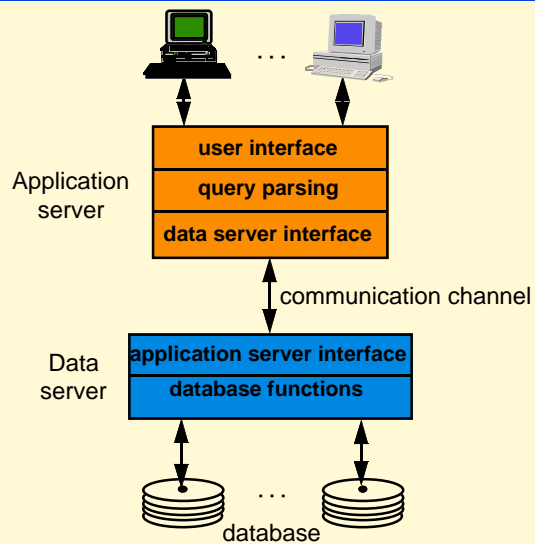
The Solution

- Increase the I/O bandwidth
 - \Rightarrow data partitioning
 - \Rightarrow parallel data access
- Origins (1980's): **database machines**
 - \Rightarrow hardware-oriented \Rightarrow bad cost-performance \Rightarrow **failure**
 - \Rightarrow notable exception : ICL's CAFS ISP
- 1990's: same solution but using standard hardware components integrated in a **multiprocessor**
 - \Rightarrow software-oriented
 - \Rightarrow standard essential to exploit continuing technology improvements

Multiprocessor Objectives

- High-performance with better cost-performance than mainframe or vector supercomputer
- Use many nodes, each with good cost-performance, communicating through network
 - good cost via high-volume components
 - good performance via bandwidth
- Trends
 - microprocessor and memory (DRAM): off-the-shelf
 - network (multiprocessor edge): custom
- The real challenge is to parallelize applications to run with good **load balancing**

Data Server Architecture



Objectives of Data Servers

Avoid the shortcomings of the traditional DBMS approach

- ➡ centralization of data and application management
- ➡ general-purpose OS (not DB-oriented)

by separating the functions between

- ➡ **application server** (host computer)
- ➡ **data server** (database computer, back-end computer)

Data Server Approach: Assessment

■ Advantages

- ➡ integrated data control by the server (black box)
- ➡ increased performance by dedicated system
- ➡ can better exploit parallelism
- ➡ fits well in distributed environments

■ Potential problems

- ➡ communication overhead between application and data server
 - ◆ high-level interface
- ➡ high cost with mainframe servers

Parallel Data Processing

■ Three ways of exploiting high-performance multiprocessor systems:

- ❶ automatically detect parallelism in sequential programs (e.g., Fortran, OPS5)
- ❷ augment an existing language with parallel constructs (e.g., C*, Fortran90)
- ❸ offer a new language in which parallelism can be expressed or automatically inferred

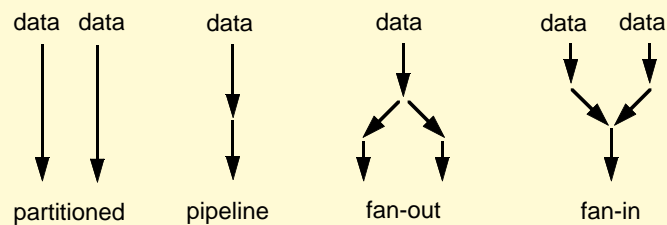
■ Critique

- ❶ hard to develop parallelizing compilers, limited resulting speed-up
- ❷ enables the programmer to express parallel computations but too low-level
- ❸ can combine the advantages of both (1) and (2)

Data-based Parallelism

■ Infer parallelism inherent in SQL queries

- ➡ inter-query
- ➡ intra-query
- ➡ intra-operation



Parallel Database Management System

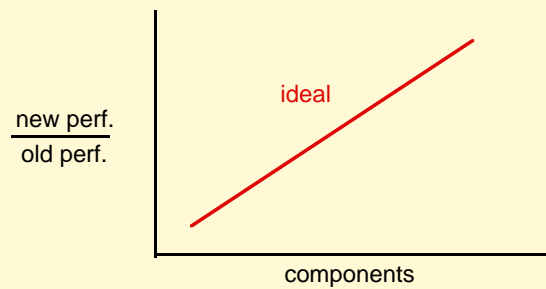
- Loose definition : a DBMS implemented on a tightly coupled multiprocessor
- Alternative extremes
 - ➡ straightforward porting of relational DBMS (the software vendor edge)
 - ➡ new hardware/software combination (the computer manufacturer edge)
- Naturally extends to distributed databases with one server per site

Parallel Database Systems - Objectives

- Much better cost / performance than mainframe solution
- High-performance through parallelism
 - ➡ high throughput with inter-query parallelism
 - ➡ low response time with intra-operation parallelism
- High availability and reliability by exploiting data replication
- Extensibility with the ideal goals
 - ➡ linear speed-up
 - ➡ linear scale-up

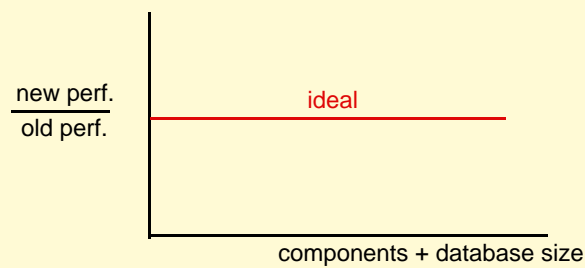
Linear Speed-up

Linear increase in performance for a constant DB size and proportional increase of the system components (processor, memory, disk)



Linear Scale-up

Sustained performance for a linear increase of database size and proportional increase of the system components.



Barriers to Parallelism

■ Startup

- the time needed to start a parallel operation may dominate the actual computation time

■ Interference

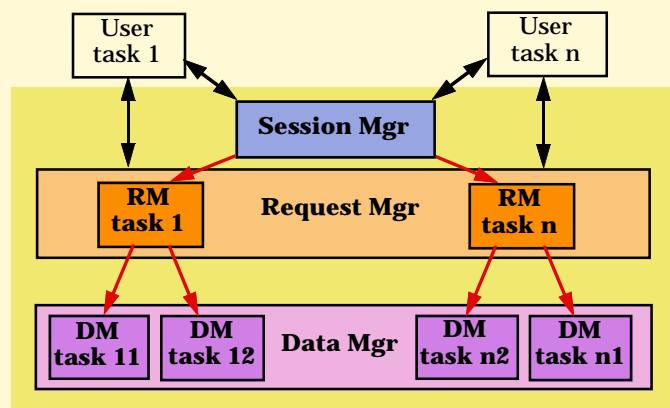
- when accessing shared resources, each new process slows down the others (hot spot problem)

■ Skew

- the response time of a set of parallel processes is the time of the slowest one

- Parallel data management techniques intend to overcome these barriers

Functional Architecture of PDBS



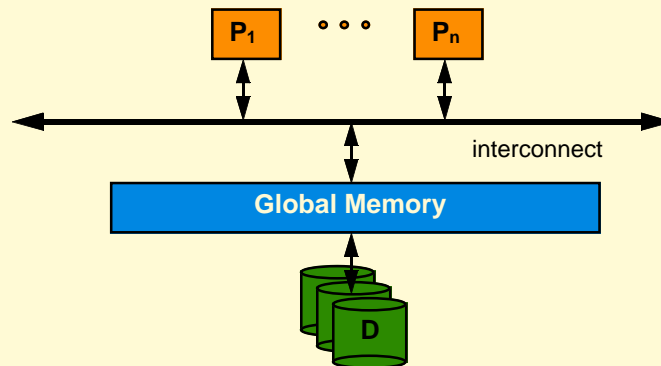
Parallel DB Functions

- session manager
 - host interface
 - transaction monitoring for OLTP
- request manager
 - compilation and optimization
 - data directory management
 - semantic data control
 - execution control
- data manager
 - execution of DB operations
 - transaction management support
 - data management

Parallel System Architectures

- Multiprocessor architecture extremes
 - shared memory (shared everything)
 - shared nothing (message-passing)
- Intermediate architecture: shared-disk

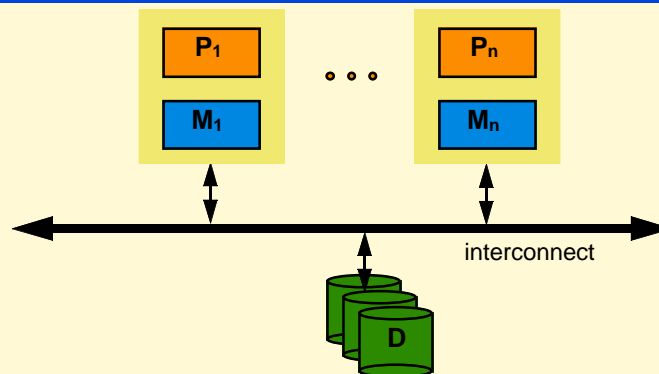
Shared-Memory Architecture



Examples: symmetric multiprocessors (Sequent, Encore, Bull's Escala), XPRS (U. of Berkeley), DBS3 (Bull)

- simplicity, load balancing, fast communication
- network cost, low extensibility, low availability

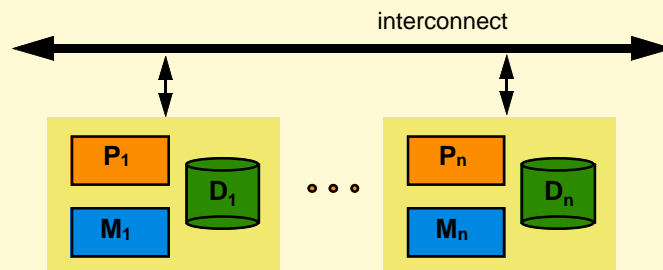
Shared-Disk Architecture



Examples : DEC's VAXcluster, IBM's IMS/VS Data Sharing

- network cost, extensibility, migration from uniprocessor
- complexity, potential performance problem for copy coherency

Shared-Nothing Architecture

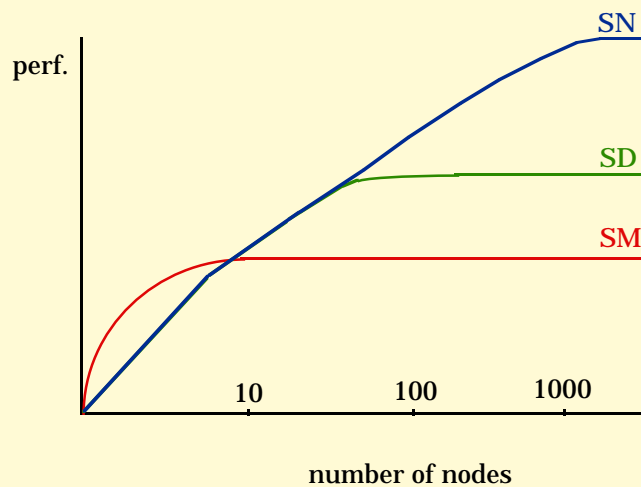


Examples : Teradata (ATT GIS), NonStopSQL (Tandem),
Gamma (U. of Wisconsin), Bubba (MCC)

■ cost, extensibility, availability

■ complexity, difficult load balancing

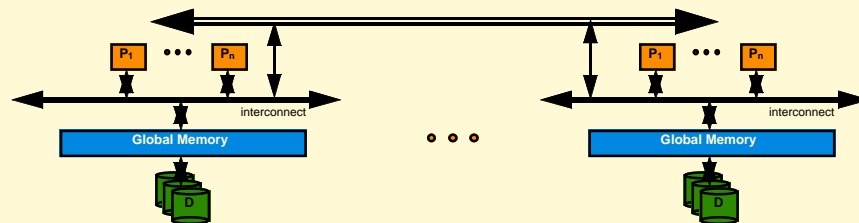
Performance Comparisons



Shared-Memory vs. Distributed Memory

- Mixes two different aspects : addressing and memory
 - addressing
 - ◆ single address space : Sequent, Encore, KSR
 - ◆ multiple address spaces : Intel, nCube
 - physical memory
 - ◆ central : Sequent, Encore
 - ◆ distributed : Intel, nCube, KSR
- Trend : single address space on distributed physical memory (KSR)
 - eases application portability
 - extensibility

Cluster of SM Nodes



- combines good load balancing of SM with extensibility of SN
- alternatives
 - limited number of large nodes, e.g., 4 x 16 processor nodes
 - high number of small nodes, e.g., 16 x 4 processor nodes, has much better cost-performance (can be a cluster of workstations)

Basic Techniques

- Data placement
 - ➡ physical placement of the DB onto multiple nodes
 - ➡ static vs. dynamic
- Parallel data processing
 - ➡ select is easy
 - ➡ join (and all other non-select operations) is more difficult
- Parallel query optimization
 - ➡ choice of the best parallel execution plans
 - ➡ automatic parallelization of the queries
- Transaction management
 - ➡ similar to distributed transaction management

Parallel Data Placement

Parallel architecture



balance the load between
all the nodes

Large volume of data



execute where the data is



data placement determines performance

Placement Alternatives

■ Clustering

- ➡ each relation entirely contained at one node
- ➡ may minimize total amount of work (if all relevant relations at the same node)

■ Declustering

- ➡ each relation horizontally fragmented across all nodes by a hash function
- ➡ may well minimize response time but increase total time

Variable Declustering

- number of nodes of a relation (home) is a function of its size and access frequency
- varies dynamically for load balancing
- physical data placement known only at query run time \Rightarrow logical data placement needed at compile time

Data Partitioning

- Each relation is divided in n partitions (subrelations), where n is a function of relation size and access frequency
- Implementation
 - ⇒ round-robin
 - ◆ maps i -th element to node $i \bmod n$
 - ◆ simple but only exact-match queries
 - ⇒ B-tree index
 - ◆ supports range queries but large index
 - ⇒ hash function
 - ◆ only exact-match queries but small index

Placement Directory

- Performs two functions
 - ⇒ f_1 (relname, placement attval) = lognode-id
 - ⇒ f_2 (lognode-id) = phynode-id
- Implementation
 - ⇒ round-robin (maps i -th elt to node $i \bmod n$): simple but only exact-match queries
 - ⇒ B-tree index: supports range queries but large index
 - ⇒ hash function: only exact-match queries but small index
- In either case, the data structure for f_1 and f_2 should be available when needed at each node

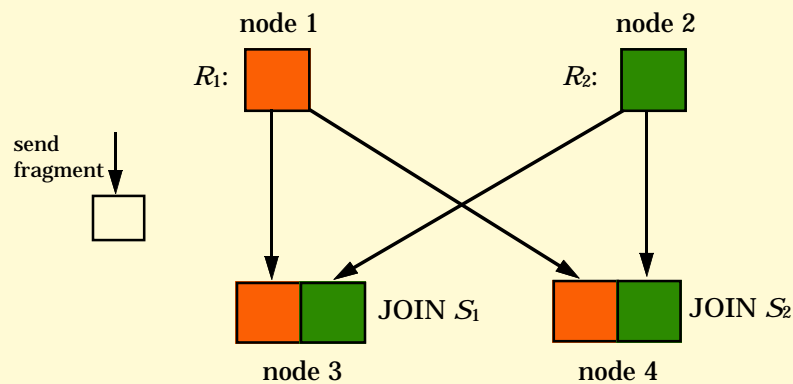
Join Processing

■ Three basic join

- ➔ parallel nested loop join: no special assumption
- ➔ parallel associative join: one relation is declustered on join attribute and equi-join
- ➔ parallel hash join: equi-join

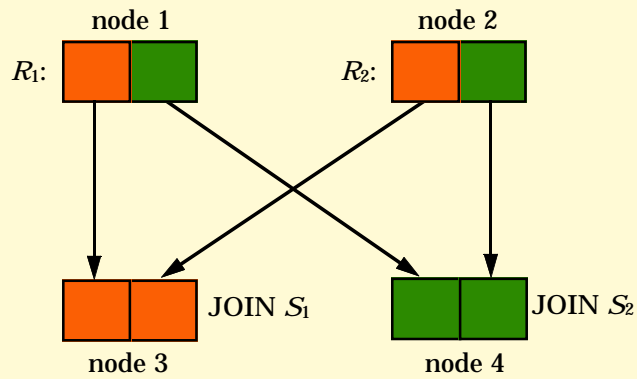
- They also apply to other complex operators such as duplicate elimination, union, intersection, etc. with minor adaptation

Parallel Nested Loop Join



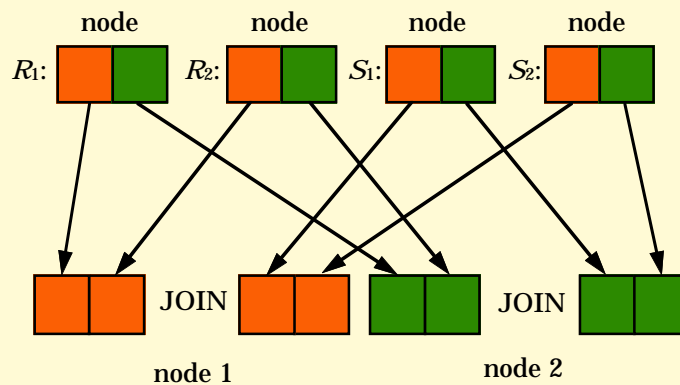
$$R \text{ JOIN } S = \text{UNION}_{i=1}^n (R \text{ JOIN } S_i)$$

Parallel Associative Join



$$R \text{ JOIN } S = \bigcup_{i=1}^n (R_i \text{ JOIN } S_i)$$

Parallel Hash Join



$$R \text{ JOIN } S = \bigcup_{i=1}^n (R \text{ JOIN } S_i)$$

Parallel Query Optimization

The objective is to select the "best" parallel execution plan for a query using the following components

Search space

- models alternative execution plans as processing trees
- left-deep vs. right-deep vs. bushy trees

Search strategy

- dynamic programming for small search space
- randomized for large search space

Cost model (abstraction of execution system)

- physical schema info. (declustering, indexes, etc.)
- statistics and cost functions

Parallel Database Systems

■ Prototypes

- EDS and DBS3 (ESPRIT)
- GAMMA (U. of Wisconsin)
- Bubba (MCC, Austin, Texas)
- XPRS (U. of Berkeley)
- GRACE (U. of Tokyo)

■ Products

- Teradata (NCR GIS)
- NonStopSQL (Tandem)
- DB2 (IBM), Oracle, Informix, Ingres, Navigator (Sybase) ...

Open Research Problems

- **Hybrid architectures**, e.g., SN composed of SM nodes
- **OS support**: using micro-kernels (Chorus, Mach)
- **Benchmarks** to stress speedup and scaleup under mixed workloads
- **Data placement** to deal with skewed data distributions and data replication
- **Parallel data languages** to specify independent and pipelined parallelism
- **Parallel query optimization** to deal with mix of precompiled queries and complex ad-hoc queries
- **Support of higher functionality** such as decutive and object capabilities