Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Semantic Data Control
- Distributed Query Processing
- Distributed Transaction Management
  - Data server approach
  - Parallel architectures
  - Parallel DBMS techniques
  - Parallel execution models
- Parallel Database Systems
- Distributed Object DBMS
- Database Interoperability
- Concluding Remarks
The Database Problem

- Large volume of data use disk and large main memory
- I/O bottleneck (or memory access bottleneck)
  - Speed(disk) << speed(RAM) << speed(microprocessor)
- Predictions
  - (Micro-) processor speed growth: 50% per year
  - DRAM capacity growth: 4× every three years
  - Disk throughput: 2× in the last ten years
- Conclusion: the I/O bottleneck worsens
The Solution

- Increase the I/O bandwidth
  - Data partitioning
  - Parallel data access

- Origins (1980's): database machines
  - Hardware-oriented bad cost-performance failure
  - Notable exception: ICL's CAFS Intelligent Search Processor

- 1990's: same solution but using standard hardware components integrated in a multiprocessor
  - Software-oriented
  - 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

Client → client interface → query parsing → data server interface → communication channel → application server interface → database functions → database → Data server

Application server

Diagram with the following components:
- Client
- Data server
- Communication channel
- Application server interface
- Database functions
- Database
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 (or host computer)
- Data server (or database computer or 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:
  1. Automatically detect parallelism in sequential programs (e.g., Fortran, OPS5)
  2. Augment an existing language with parallel constructs (e.g., C*, Fortran90)
  3. Offer a new language in which parallelism can be expressed or automatically inferred

- Critique
  1. Hard to develop parallelizing compilers, limited resulting speed-up
  2. Enables the programmer to express parallel computations but too low-level
  3. Can combine the advantages of both (1) and (2)
Data-based Parallelism

- **Inter-operation**
  - $p$ operations of the same query in parallel

- **Intra-operation**
  - the same operation in parallel on different data partitions
Parallel DBMS

- 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 DBMS - 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.

- new perf.
- old perf.

components + database size
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**
Parallel DBMS – Functional Architecture

```
User task 1

Session Mgr

RM task 1  Request Mgr  RM task n

DM task 11  DM task 12  Data Mgr  DM task n2  DM task n1

User task n
```
Parallel DBMS 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 alternatives
  - Shared memory (shared everything)
  - Shared disk
  - Shared nothing (message-passing)

- Hybrid architectures
  - Hierarchical (cluster)
  - Non-Uniform Memory Architecture (NUMA)
Shared-Memory Architecture

Examples: DBMS on symmetric multiprocessors (Sequent, Encore, Sun, etc.)
- Simplicity, load balancing, fast communication
- Network cost, low extensibility
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 (NCR), NonStopSQL (Tandem-Compaq), Gamma (U. of Wisconsin), Bubba (MCC)

- Extensibility, availability
- Complexity, difficult load balancing
Hierarchical Architecture

- 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)
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

- NUMA: single address space on distributed physical memory
  - Eases application portability
  - Extensibility
NUMA Architectures

- **Cache Coherent NUMA (CC-NUMA)**
  - statically divide the main memory among the nodes

- **Cache Only Memory Architecture (COMA)**
  - convert the per-node memory into a large cache of the shared address space
COMA Architecture

Hardware shared virtual memory

Disk

P_1

Cache Memory

P_2

Cache Memory

... 

P_n

Cache Memory
Parallel DBMS 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 and load balancing

- **Transaction management**
  - Similar to distributed transaction management
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 $i \mod n$
    - Simple but only exact-match queries
  - **B-tree index**
    - Supports range queries but large index
  - **Hash function**
    - Only exact-match queries but small index
Partitioning Schemes

- **Round-Robin**
  - Objects are distributed uniformly across machines.

- **Hashing**
  - Objects are assigned to machines based on the hash value of the object's key.

- **Interval**
  - Objects are grouped into intervals, and each interval is stored on a separate machine.

```plaintext
a-g  h-m  ...  u-z
```

Diagram:}

```
```

---

**Distributed DBMS**

© 1998 M. Tamer Özsu & Patrick Valduriez

Page 13.27
Replicated Data Partitioning

- High-availability requires data replication
  - simple solution is mirrored disks
    - hurts load balancing when one node fails
  - more elaborate solutions achieve load balancing
    - interleaved partitioning (Teradata)
    - chained partitioning (Gamma)
### Interleaved Partitioning

<table>
<thead>
<tr>
<th>Node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary copy</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>R4</td>
</tr>
<tr>
<td>Backup copy</td>
<td>r 2.3</td>
<td>r 1.1</td>
<td>r 1.2</td>
<td>r 1.3</td>
</tr>
<tr>
<td></td>
<td>r 3.2</td>
<td>r 3.2</td>
<td>r 2.1</td>
<td>r 2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>r 3.1</td>
<td></td>
</tr>
</tbody>
</table>
### Chained Partitioning

<table>
<thead>
<tr>
<th>Node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary copy</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>R4</td>
</tr>
<tr>
<td>Backup copy</td>
<td>r4</td>
<td>r1</td>
<td>r2</td>
<td>r3</td>
</tr>
</tbody>
</table>
Placement Directory

- Performs two functions
  - $F_1$ (relname, placement attribute) = lognode-id
  - $F_2$ (lognode-id) = phynode-id

- In either case, the data structure for $f_1$ and $f_2$ should be available when needed at each node
Join Processing

- Three basic algorithms for intra-operator parallelism
  - 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

\[ R \bowtie S \rightarrow \bigcup_{i=1,n} (R \bowtie S_i) \]
Parallel Associative Join

\[ R \bowtie S \rightarrow \bigcup_{i=1,n} (R_i \bowtie S_i) \]
Parallel Hash Join

\[ R \bowtie S \rightarrow \bigcup_{i=1, P} (R_i \bowtie S_i) \]
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 operator 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. (partitioning, indexes, etc.)
- Statistics and cost functions
Execution Plans as Operators Trees

Left-deep

Right-deep

Zig-zag

Bushy
Equivalent Hash-Join Trees with Different Scheduling
Load Balancing

- Problems arise for intra-operator parallelism with *skewed* data distributions
  - attribute data skew (AVS)
  - tuple placement skew (TPS)
  - selectivity skew (SS)
  - redistribution skew (RS)
  - join product skew (JPS)

- Solutions
  - sophisticated parallel algorithms that deal with skew
  - dynamic processor allocation (at execution time)
Data Skew Example
Some Parallel DBMSs

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

- **Products**
  - Teradata (NCR)
  - NonStopSQL (Tandem-Compac)
  - DB2 (IBM), Oracle, Informix, Ingres, Navigator (Sybase) ...
Open Research Problems

- Hybrid architectures
- OS support: using micro-kernels
- 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 rules and objects