Outline

- Introduction & architectural issues
- Data distribution
- Distributed query processing
- Distributed query optimization
- Distributed transactions & concurrency control
- Distributed reliability
- Data replication
- Parallel database systems
  - Data placement and query processing
  - Database clusters
- Database integration & querying
- Peer-to-Peer data management
- Stream data management
- MapReduce-based distributed data management

Parallelization Motivation

- Large volume of data ⇒ use disk and large main memory
- I/O bottleneck (or memory access bottleneck)
  - Speed(disk) << speed(RAM) << speed(microprocessor)
- Increase the I/O bandwidth
  - Data partitioning
  - Parallel data access

History

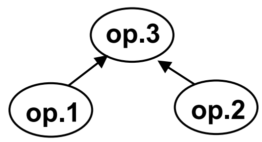
- Origins (1980's): *database machines*
  - Hardware-oriented ➡ bad cost-performance ➡ failure
  - Notable exception : ICL's CAFS Intelligent Search Processor
- 1990's: same solution but using standard hardware components integrated in a multiprocessor
  - Software-oriented
  - Standard essential to exploit continuing technology improvements

Multiprocessor Objectives

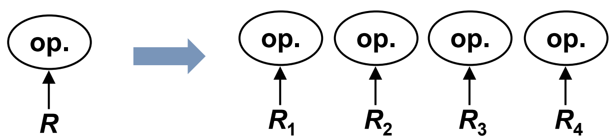
- High-performance with better cost-performance than mainframe or vector supercomputer
- Use many nodes, each with good cost-performance, communicating through network
  - Good cost via high-volume components
  - Good performance via bandwidth
- Trends
  - Microprocessor and memory (DRAM): off-the-shelf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

Data-based Parallelism

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



- Intra-operation
  - The same operation in parallel



Parallel DBMS

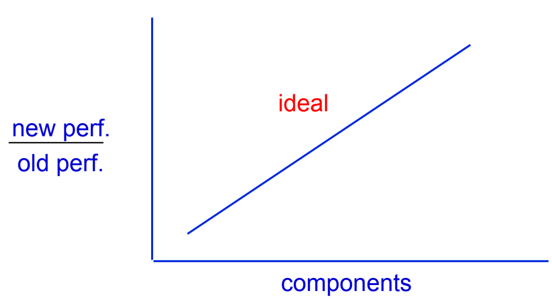
- Loose definition: a DBMS implemented on a tighly coupled multiprocessor
- Alternative extremes
  - Straighforward porting of relational DBMS (the software vendor edge)
  - New hardware/software combination (the computer manufacturer edge)
- Naturally extends to distributed databases with one server 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

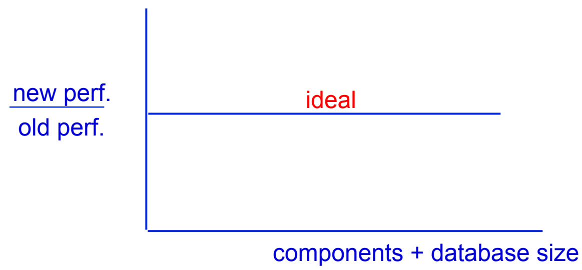
Linear Speed-up

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



Linear Scale-up

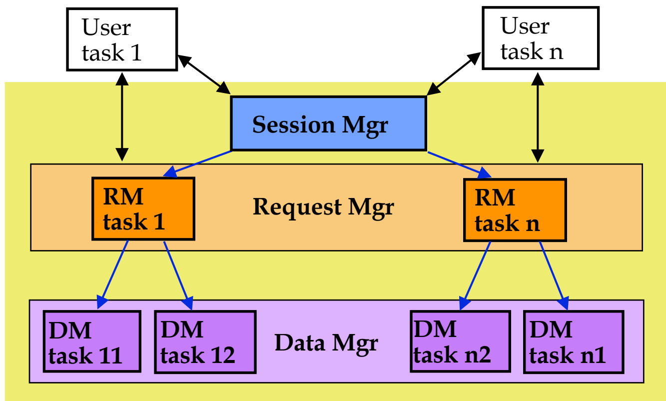
Sustained performance for a linear increase of database size and proportional increase of the system components.



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



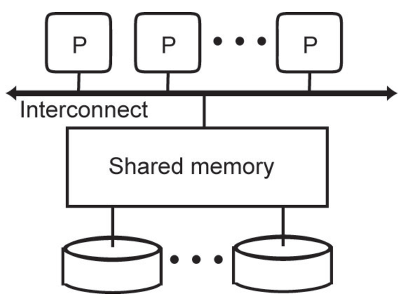
Parallel DBMS Functions

- 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

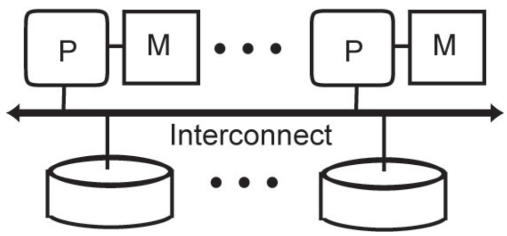
- Multiprocessor architecture alternatives
  - Shared memory (SM)
  - Shared disk (SD)
  - Shared nothing (SN)
- Hybrid architectures
  - Non-Uniform Memory Architecture (NUMA)
  - Cluster

Shared-Memory



DBMS on symmetric multiprocessors (SMP)
Prototypes: XPRS, Volcano, DBS3
+ Simplicity, load balancing, fast communication
- Network cost, low extensibility

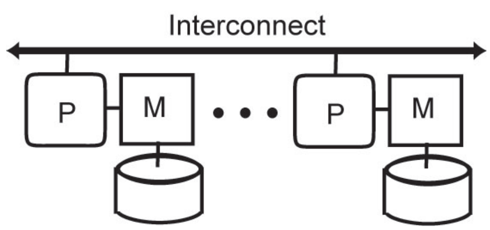
Shared-Disk



Origins : DEC's VAXcluster, IBM's IMS/VS Data Sharing
Used first by Oracle with its Distributed Lock Manager
Now used by most DBMS vendors

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

Shared-Nothing



Used by Teradata, IBM, Sybase, Microsoft for OLAP
Prototypes: Gamma, Bubba, Grace, Prisma, EDS

+ Extensibility, availability
- Complexity, difficult load balancing

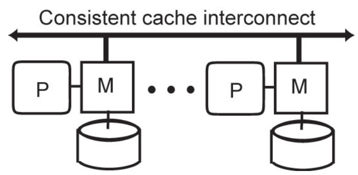
Hybrid Architectures

- Various possible combinations of the three basic architectures are possible to obtain different trade-offs between cost, performance, extensibility, availability, etc.
- Hybrid architectures try to obtain the advantages of different architectures:
  - efficiency and simplicity of shared-memory
  - extensibility and cost of either shared disk or shared nothing
- 2 main kinds: NUMA and cluster

NUMA

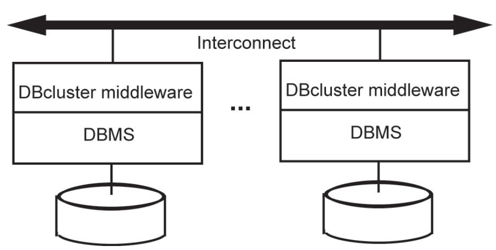
- Shared-Memory vs. Distributed Memory
  - Mixes two different aspects : addressing and memory
    - Addressing: single address space vs multiple address spaces
    - Physical memory: central vs distributed
- NUMA = single address space on distributed physical memory
  - Eases application portability
  - Extensibility
- The most successful NUMA is Cache Coherent NUMA (CC-NUMA)

CC-NUMA



- Principle
  - Main memory distributed as with shared-nothing
  - However, any processor has access to all other processors' memories
- Similar to shared-disk, different processors can access the same data in a conflicting update mode, so global cache consistency protocols are needed.
  - Cache consistency done in hardware through a special consistent cache interconnect
    - Remote memory access very efficient, only a few times (typically between 2 and 3 times) the cost of local access

Cluster



- Combines good load balancing of SM with extensibility of SN
- Server nodes: off-the-shelf components
  - From simple PC components to more powerful SMP
  - Yields the best cost/performance ratio
  - In its cheapest form,
- Fast standard interconnect (e.g., Myrinet and Infiniband) with high bandwidth (Gigabits/sec) and low latency

SN cluster vs SD cluster

- SN cluster can yield best cost/performance and extensibility
  - But adding or replacing cluster nodes requires disk and data reorganization
- SD cluster avoids such reorganization but requires disks to be globally accessible by the cluster nodes
  - Network-attached storage (NAS)
    - distributed file system protocol such as NFS, relatively slow and not appropriate for database management
  - Storage-area network (SAN)
    - Block-based protocol thus making it easier to manage cache consistency, efficient, but costlier

Discussion

- For a small configuration (e.g., 8 processors), SM can provide the highest performance because of better load balancing
- Some years ago, SN was the only choice for high-end systems. But SAN makes SN a viable alternative with the main advantage of simplicity (for transaction management)
  - SD is now the preferred architecture for OLTP
  - But for OLAP databases that are typically very large and mostly read-only, SN is used
- Hybrid architectures, such as NUMA and cluster, can combine the efficiency and simplicity of SM and the extensibility and cost of either SD or SN

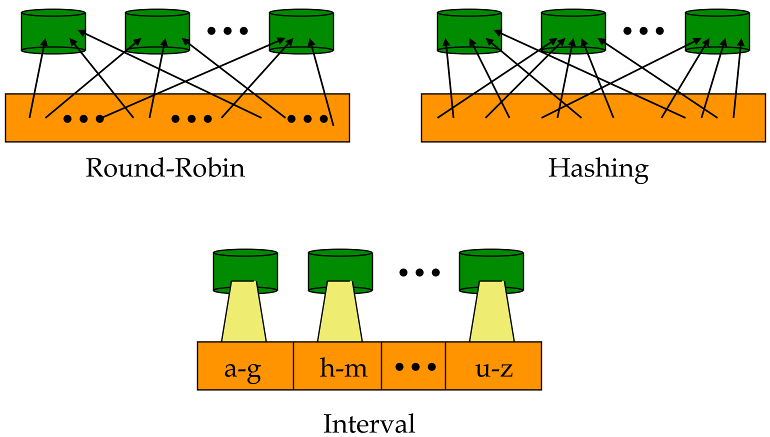
Parallel DBMS Techniques

- 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



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

Node	1	2	3	4
Primary copy	R_1	R_2	R_3	R_4
Backup copy		$r_{1.1}$	$r_{1.2}$	$r_{1.3}$
	$r_{2.3}$		$r_{2.1}$	$r_{2.2}$
	$r_{3.2}$	$r_{3.2}$		$r_{3.1}$

Chained Partitioning

Node	1	2	3	4
Primary copy	R_1	R_2	R_3	R_4
Backup copy	r_4	r_1	r_2	r_3

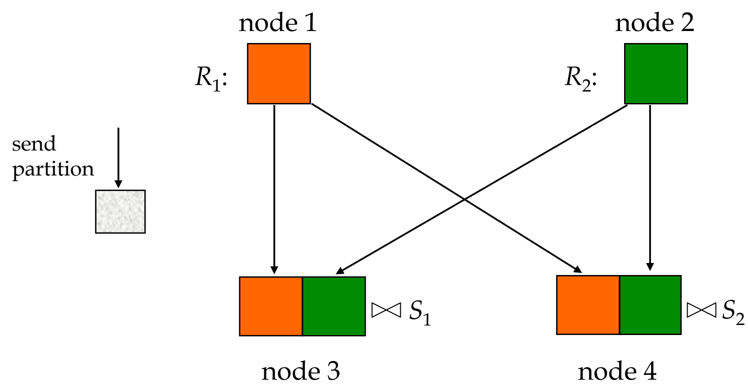
Placement Directory

- Performs two functions
  - F_1 (relname, placement attval) = 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

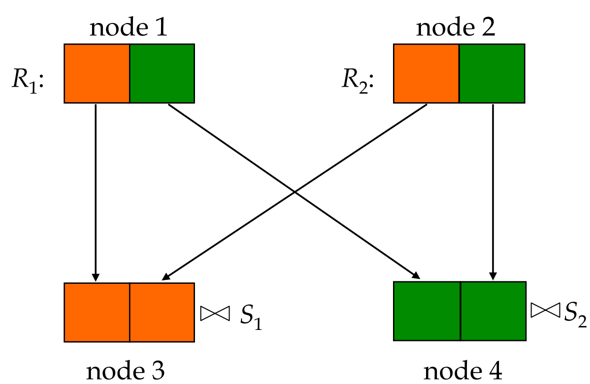
- Three basic algorithms for intra-operator parallelism
  - Parallel nested loop join: no special assumption
  - Parallel associative join: one relation is declustered on join attribute and equi-join
  - Parallel hash join: equi-join
- They also apply to other complex operators such as duplicate elimination, union, intersection, etc. with minor adaptation

Parallel Nested Loop Join



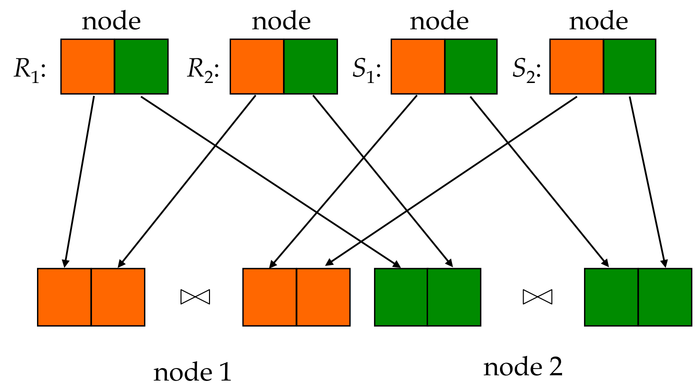
$$R \bowtie S = \bigcup_{i=1}^{n} (R \bowtie S_i)$$

Parallel Associative Join



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

Parallel Hash Join



$$R \bowtie S = \bigcup_{i=1}^{p} (R_i \bowtie S_i)$$

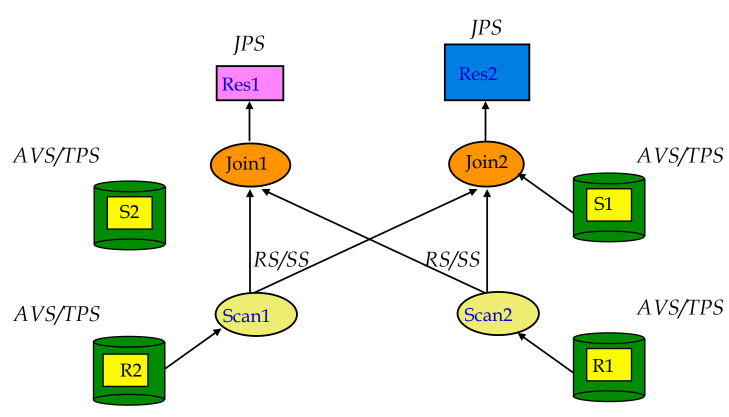
Parallel Query Optimization

- 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

Load Balancing

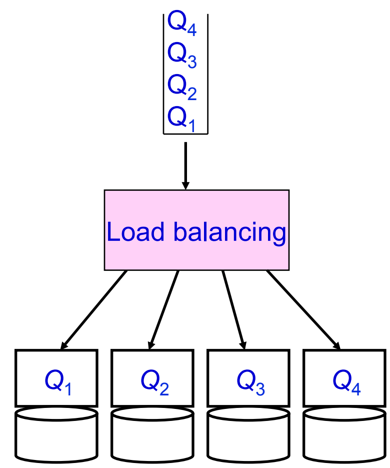
- Problems arise for intra-operator parallelism with *skewed* data distributions
  - attribute data skew (AVS)
  - tuple placement skew (TPS)
  - selectivity skew (SS)
  - redistribution skew (RS)
  - join product skew (JPS)
- Solutions
  - sophisticated parallel algorithms that deal with skew
  - dynamic processor allocation (at execution time)

Data Skew Example



Load Balancing in a DB Cluster

- Choose the node to execute Q
  - round robin
  - The least loaded
    - Need to get load information
- Fail over
  - In case a node N fails, N's queries are taken over by another node
    - Requires a copy of N's data or SD
- In case of interference
  - Data of an overloaded node are replicated to another node



Load Balancing in a DB Cluster

- Choose the node to execute Q
  - round robin
  - The least loaded
    - Need to get load information
- Fail over
  - In case a node N fails, N's queries are taken over by another node
    - Requires a copy of N's data or SD
- In case of interference
  - Data of an overloaded node are replicated to another node