

Parallel and Distributed Databases

UA.DETI.CBD
José Luis Oliveira / Carlos Costa

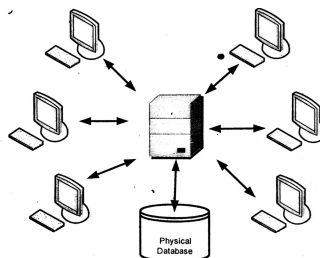


1

Centralized DBMS

❖ Centralized database (DBMS)

- Data is located in one place (e.g. server)
- All DBMS functionalities are done by the server
 - Enforcing ACID properties of transactions
 - Concurrency control, recovery mechanisms
 - Answering queries



2

2

Biggest Database Problem

- ❖ Large volume of data \Rightarrow use disk and large memory
- ❖ **Bottlenecks**
 - Speed(disk) \ll speed(RAM) \ll speed(microprocessor)
- ❖ Evolution
 - Processor speed growth (with multicore): 50 % per year
 - DRAM capacity growth: 4 \times every three years
 - Disk throughput: 2 \times in the last ten years
- ❖ **Biggest bottleneck: I/O**
- ❖ Solution to increase the I/O bandwidth
 - parallel data access
 - data partitioning



3

3

Parallel and Distributed DBMS

- ❖ **Parallel** database (Parallel DBMS)
 - A "Centralized" DB with multiple resources such as CPUs and disks in parallel.
 - It supports parallel operations such as query processing and data loading.
- ❖ **Distributed** databases (DDBMS)
 - Data is stored in multiple places (each is running a DBMS)
 - New notion of distributed transactions
 - DBMS functionalities are now distributed over many machines



4

4

Why Parallel Databases?

- ❖ Processing 1 Terabyte?
 - at 10MB/s => ~1.2 days to scan
 - 1000 x parallel => 1.5 minute to scan
- ❖ **Divide a big problem** into many smaller ones to be solved in parallel
- ❖ Data may be stored in a distributed fashion
 - But the distribution is governed solely by **performance** considerations.
- ❖ **Large-scale parallel database** systems increasingly used for:
 - storing large volumes of data
 - processing time-consuming decision-support queries
 - providing high throughput for transaction processing



5

5

Why Distributed Databases?

- ❖ **Scalability**
 - If data volume, read load or write load grows bigger than a single machine can handle, you can potentially **spread the load across multiple machines**.
- ❖ **Fault tolerance / High availability**
 - Multiple machines can provide **redundancy**. When one fails, another one can take over.
- ❖ **Latency**
 - Applications are by nature distributed. With users around the world, and DB servers at various locations worldwide, **users can be served from a closer datacenter**.



6

6

Parallel Databases

- ❖ Parallel databases improve processing and I/O speeds by using **multiple CPUs and disks in parallel**
 - data can be partitioned across multiple disks
 - each processor can work independently on its own partition
- ❖ Exploit the parallelism in data management in order to deliver **high-performance, high-availability** and **extensibility**
 - support very large databases with very high loads
- ❖ Different **queries** can be **run in parallel**.



7

7

Parallel Databases (cont.)

- ❖ **Critical issues**
 - data placement
 - parallel query processing
 - load balancing
- ❖ **Research** done in the **context** of the **relational model** provides a good basis for data-based parallelism
 - individual relational operations (e.g., sort, join, aggregation) can be executed in parallel



8

8

Parallel vs Distributed Databases

Although the **basic principles** of **parallel DBMS** are the **same** as in **distributed DBMS**, the **techniques are** fairly **different**

typically...

Parallel DB

- ❖ Fast interconnect
- ❖ Homogeneous software
- ❖ High performance is goal
- ❖ Transparency is goal

Distributed DB

- ❖ Geographically distributed
- ❖ Data sharing is goal
- ❖ Disconnected operation possible



9

9

Parallel vs Distributed Databases

❖ Distributed processing can use parallel processing

- Parallel processing on a single machine (not the opposite)

❖ Parallel Databases

- Machines are physically close (e.g. same server room)
- .. and connects with dedicated high-speed LANs
- Communication cost is assumed to be small
- Architecture: can be **shared-memory**, **shared-disk** or **shared-nothing**

❖ Distributed Databases

- Machines can be in distinct geographic locations
- .. and connected using public-purpose network, e.g., Internet
- Communication cost and problems cannot be ignored
- Architecture: usually **shared-nothing**



10

10

Parallel DBMS – Main Goals

- ❖ **High-performance** through **parallelization** of various operations
 - **High throughput** with inter-query parallelism
 - **Low response time** with intra-operation parallelism
 - **Load balancing** is the ability of the system to divide a given workload equally among all processors
- ❖ **High availability** by exploiting **data replication**
- ❖ **Extensibility** with the ideal goals
 - Linear speed-up
 - Linear scale-up

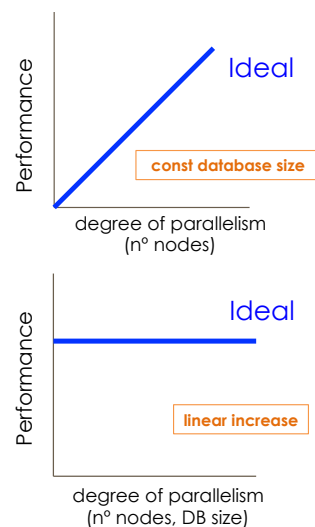


11

11

Ideal Extensibility Scenario

- ❖ **Speed-Up**
 - refers to a **linear increase** in **performance** for a **constant database size** while the **number of nodes** (i.e. processing and storage power) are **increased linearly**
 - more resources means proportionally less time for given amount of data
- ❖ **Scale-Up**
 - refers to a **sustained performance** for a **linear increase** in both **database size** and **number of nodes**
 - if resources increased in proportion to increase in data size, time is constant



12

12

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



13

13

Database Architectures

... to scale to higher loads:

❖ Multiprocessor architecture

- Shared memory (SM) Also called vertical scaling (or scaling up)
Simplest approach - buy a more powerful machine
- Shared disk (SD)
- Shared nothing (SN) Aka horizontal scaling (or scaling out)

❖ Hybrid architectures

- Non-Uniform Memory Architecture (NUMA)
- Cluster (or hierarchical)



14

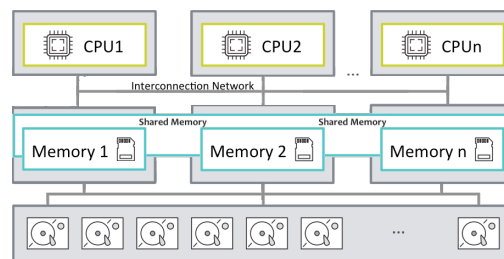
14

Shared Memory

- ❖ Multiple processors share the main memory (RAM) space, but each processor has its own disk (HDD)
 - provide communications among them and avoid redundant copies

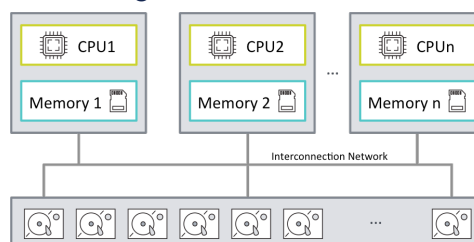
❖ Bottlenecks

- cost is super-linear: a machine with twice resources (CPU, RAM, disk) typically costs significantly more than twice
- a machine twice the size cannot necessarily handle twice the load
- limited fault tolerance



Shared Disk

- ❖ Uses several machines with independent CPUs and RAM, but **stores data on an array of disks** that is shared between the machines, connected via a fast network
- ❖ Used for some data warehousing workloads
- ❖ **Advantages** over shared memory
 - each processor has its own memory - is not a bottleneck
 - a simple way to provide a degree of fault tolerance.
- ❖ **Disadvantages**
 - I/O contention
 - limited scalability



Shared Nothing

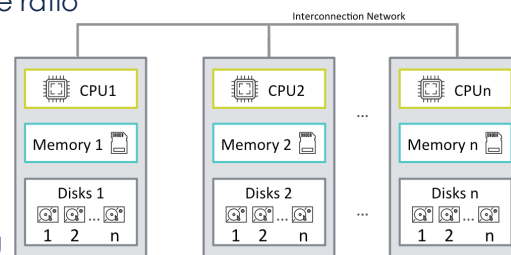
- ❖ Each machine running the database software (**node**) uses its CPUs, RAM and disks independently
- ❖ Any coordination between nodes is done at the software level, using a conventional network
- ❖ Most common architecture nowadays

- ❖ **Advantages:**

- best price/performance ratio
- extensibility
- availability
- reduce latency

- ❖ **Disadvantages:**

- 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**
- ❖ Two main types:
 - NUMA (*non-uniform memory access*)
 - Cluster

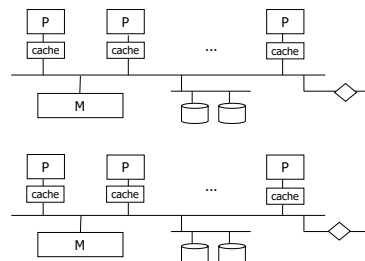
NUMA (non-uniform memory access)

- ❖ Shared Memory vs. Distributed Memory
 - mixes two different aspects:
 - addressing: single address space *and* multiple address spaces
 - physical memory: central *and* distributed
- ❖ NUMA uses **single address space on distributed physical memory**
 - eases application portability
 - extensibility
- ❖ Cache Coherent NUMA (CC-NUMA)
 - the most successful

19

CC-NUMA

- ❖ Principle: main **memory distributed** as with shared-nothing. However, any **processor has access to all other processors' memories**
 - remote memory access very efficient, only a few times (typically between 2 and 3 times) the cost of local access
- ❖ Different processors can access the same data in a conflicting update mode
 - a global cache consistency protocols are needed



20

Parallel & Distributed DBMS Techniques

❖ Data placement

- Physical placement of the DB into multiple nodes
- Static vs. Dynamic

❖ Parallel data processing algorithms

- 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

❖ Distributed Transaction management



21

21

Distributed Data Storage

Two common ways of distribute data across nodes:

❖ Partitioning

- splitting a big database into smaller subsets called partitions
- different partitions can be assigned to different nodes

❖ Replication

- keeping a copy of the same data on several different nodes; potentially in different locations
- provides redundancy; if some nodes are unavailable, the data can still be served from the remaining nodes
- can also help improve performance

❖ Replication and Partitioning can be combined



22

22

Data Transparency

- ❖ Transparency means that the DDBMS hides all the added complexities of distribution
 - allowing users to think that they are working with a single centralised system.
- ❖ Consider transparency issues in relation to:
 - Replication mechanism
 - Partitioning mechanism
 - Location



23

23

I/O Parallelism

- ❖ In horizontal partitioning, tuples of a relation are divided among many disks

Partitioning techniques (number of disks = n):

- ❖ **Round-robin**
 - send the i^{th} tuple inserted in the relation to the disk: $i \bmod n$.
- ❖ **Hash partitioning**
 - apply a hash function to one or more attributes that range $0 \dots n - 1$
- ❖ **Range partitioning**
 - associates a range of key attribute(s) to every partition



24

24

Comparison of Partitioning Techniques

- ❖ Evaluate how well partitioning techniques support the following types of data access:
 - Scanning the entire relation.
 - Locating a tuple associatively – point queries.
 - E.g., $r.A = 25$.
 - Locating all tuples such that the value of a given attribute lies within a specified range – range queries.
 - E.g., $10 \leq r.A < 25$.

	Round Robin	Hashing	Range
Sequential Scan	Best/good parallelism	Good	Good
Point Query	Difficult	Good for hash key	Good for range vector
Range Query	Difficult	Difficult	Good for range vector

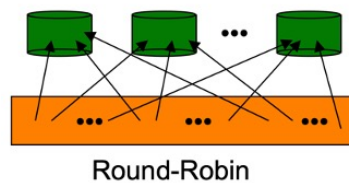


25

25

Round robin

- ❖ Advantages
 - Best suited for sequential scan of entire relation on each query.
 - All disks have almost an equal number of tuples; retrieval work is thus well balanced between disks.
- ❖ Location and Range queries are difficult to process
 - No clustering – tuples are scattered across all disks

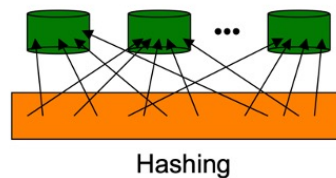


26

26

Hash partitioning

- ❖ Good for sequential access
 - Assuming hash function is good, and partitioning attributes form a key, tuples will be equally distributed between disks.
 - Retrieval work is then well balanced between disks.
- ❖ Good for point queries on partitioning attribute
 - Can lookup single disk, leaving others available for answering other queries.
 - Index on partitioning attribute can be local to disk, making lookup and update more efficient
- ❖ No clustering, so difficult to answer range queries

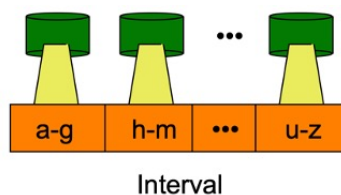


27

27

Range partitioning

- ❖ Good for sequential access
- ❖ Provides data clustering by partitioning attribute value.
- ❖ Good for point queries on partitioning attribute: only one disk needs to be accessed.
- ❖ For range queries on partitioning attribute, one to a few disks may need to be accessed
 - Remaining disks are available for other queries.
 - Good if result tuples are from one to a few blocks.



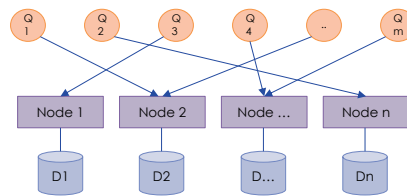
28

28

Query Parallelism

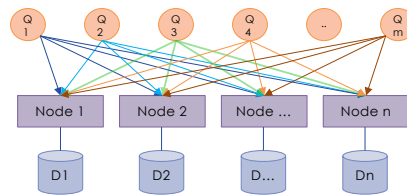
❖ Interquery Parallelism

- Parallel execution of multiple queries generated by concurrent transactions



❖ Intraquery Parallelism

- Split the execution of a single query in parallel on multiple nodes



29

29

Interquery Parallelism

❖ To **increase** the **transactional throughput**

- used primarily to scale up a transaction processing system to support a **larger number of transactions per second**

❖ **Easiest** form of parallelism to support

- particularly in a **shared-memory** parallel database

❖ More **complicated** on **shared-disk** or **shared-nothing**

- locking and logging must be coordinated by passing messages between processors
- data in a local buffer may have been updated at another processor
- cache-coherency has to be maintained: reads and writes of data in buffer must find latest version of data



30

30

Intraquery Parallelism

- ❖ The **same query** is **executed** by **many processors**, **each** one **working** on a **subset** of the **data**
 - for speeding up long-running queries
- ❖ Two complementary forms of intraquery parallelism:
 - **Intra-operation:**
 - break up a query into multiple parts within a single database partition and execute these parts at the same time.
 - **Inter-operation:**
 - break up a query into multiple parts across multiple partitions of a partitioned database on a single server or between multiple servers
- ❖ Intra-operation scales better with increasing parallelism because the number of tuples processed by each operation is typically more than the number of operations in a query

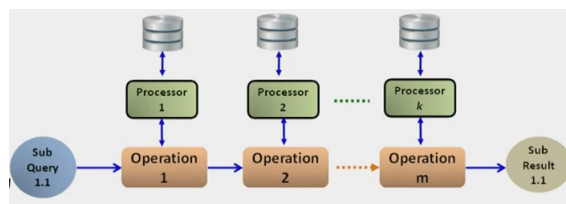


31

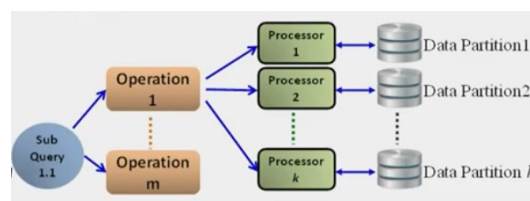
31

IntraQuery Operator Parallelism

- ❖ **Inter-operator (Pipeline):** ordered (or partially ordered) tasks and **different machines** are **performing different tasks**



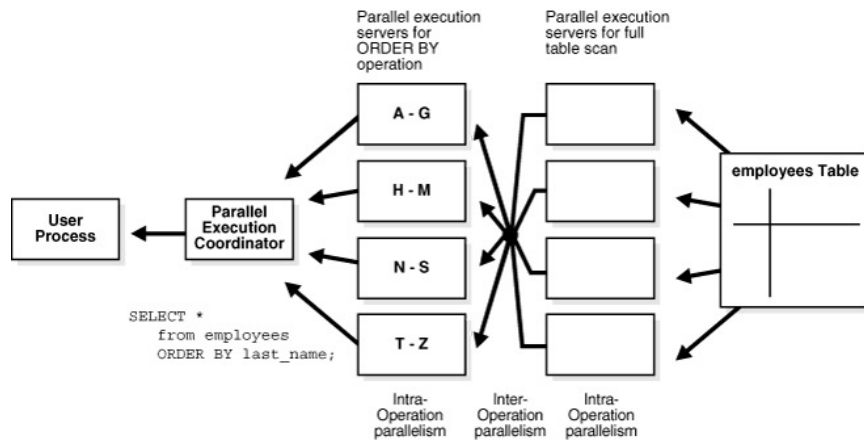
- ❖ **Intra-operator (Partitioned):** a **task divided** over all machines to **run in parallel**



32

32

Intraquery Parallelism



33

33

Parallel Data Processing

The following discussion assumes:

- ❖ **Read-only** queries
- ❖ **Shared-nothing** architecture
 - shared-nothing architectures can be efficiently simulated on shared-memory and shared-disk systems
- ❖ **n processors** (P_0, \dots, P_{n-1}) and **n disks** (D_0, \dots, D_{n-1}) where disk D_i is associated with processor P_i
 - if a processor has multiple disks they can simply simulate a single disk D_i
- ❖ We will focus on **sort**, **select** and **join** operators
 - other binary operators (such as union) can be handled in similar way to join



34

34

Parallel Selection - $\sigma_c(R)$

- ❖ Relation **R** is **partitioned** over **m machines**
 - each partition of R is around $|R|/m$ tuples
- ❖ **Each machine scans** its own **partition** and applies the **Selection** condition **c**
- ❖ **Data Partitioning impact:**
 - **round robin** or a **hash function** (over the entire tuple)
 - relation is expected to be well distributed over all nodes
 - **all partitions will be scanned**
 - **range** or hash-based (on the selection column)
 - relation can be clustered on few nodes
 - **few partitions need to be touched**



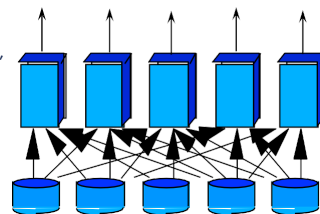
35

35

Parallel Sorting

1. Range-Partitioning Sort

- Choose processors P_0, \dots, P_m , where $m \leq n-1$ to do sorting
- Re-partition R based on ranges (on the sorting attributes) into m partitions
 - this step **requires I/O** and **communication overhead**
- Machine i receives all i^{th} partitions from all machines and sort that partition, without any interaction with the others
 - P_i stores the tuples it received temporarily on disk D_i
- **Final merge** operation is trivial
 - range-partitioning ensures that, for $i < j < m$, the key values in processor P_i are all less than the key values in P_j
- Skewed data is an issue
 - ranges can be of different width
 - apply sampling phase first



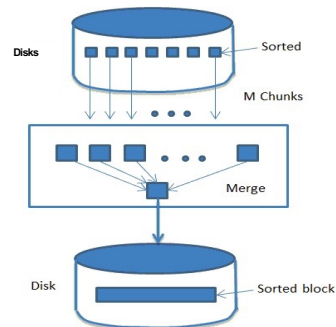
36

36

Parallel Sorting (Cont.)

2. Parallel External Sort-Merge

- Assume the relation has already been partitioned among disks D_0, \dots, D_{n-1} (in whatever manner).
- **Each node sorts its own data**
- **All nodes start sending their sorted data** (one block at a time) to a **single machine**
- This **machine** applies **merge-sort** technique as data come

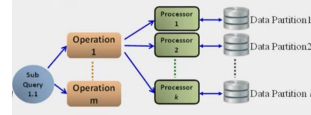


Parallel Join

- ❖ The join operation **requires pairs** of **tuples** to be **tested** to see if they satisfy the join condition
 - If tuples satisfy the join condition, the pair is added to the join output
- ❖ Steps...
 1. **Parallel join algorithms** attempt to **split the pairs-testing** over **several processors**
 2. **Each processor** then **computes part** of the **join** locally
 3. **Results** from each processor are **collected** together to **produce** the **final result**

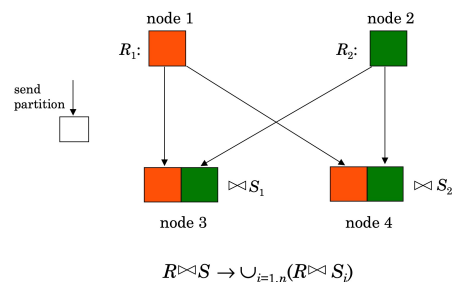
Join Algorithms

- ❖ Three basic parallel join algorithms for partitioned databases
 - **Parallel Nested Loop (PNL)**
 - **Parallel Associative Join (PAJ)**
 - **Parallel Hash Join (PHJ)**
- ❖ All previous **algorithms** are **intra-operator parallelism**
- ❖ They also apply to other complex operators such as duplicate elimination, union, intersection, etc. with minor adaptation
- ❖ Next Examples:
 - join of two **relations R** and **S** that are partitioned over **m** and **n nodes**, respectively



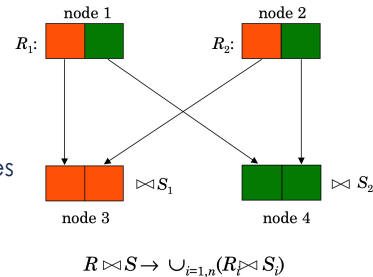
Parallel Nested Loop Join

- ❖ **Cartesian product $R \times S$** of relations **R** and **S** , in parallel.
 - **Simplest** and most **general** method
- ❖ **Algorithm phases:**
 1. each fragment of R (outer relation) is send and compared to each fragment of S (inner relation)
 - this phase is done in parallel by m nodes
 2. each S -node j receives relation R entirely, and locally joins R with fragment S_j .
 - join processing may start as soon as data are received
- ❖ **Optimization:**
 - $R \ll S$
 - If S has **index** on the join attribute at each partition, this is pretty fast



Parallel Associative Join

- ❖ Applies **only** to **equijoin** with **one** of the **operand relations partitioned** according to the **join attribute**
- ❖ Assume
 - **equijoin** predicate is on **attribute A** from **R**, and **B** from **S**
 - **S** is **partitioned** according to **hash function applied to attribute B**
 - tuples of **S** that have the same $h(B)$ value are placed at the same node
 - **no knowledge** of how **R** is **partitioned**
- ❖ Algorithm phases:
 1. relation **R** is **sent** associatively to the **S-nodes** based on the **hash function h** applied to **attribute A**
 2. **each S-node j** receives in parallel from the different **R-nodes** the relevant subset of **R** (i.e., R_j) and **joins** it **locally** with the fragments S_j



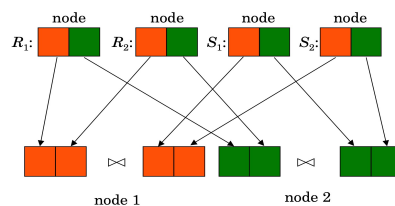
41

41

Parallel Hash Join

- ❖ Generalization of parallel associative join algorithm
 - Does not require any specific partitioning of the operand relations
- ❖ **Basic idea:**
 - **partition** of **R** and **S** into the same number distinct **p fragments**
 - R_1, R_2, \dots, R_p , and S_1, S_2, \dots, S_p ,
 - The **p nodes** may be **selected** at **run time** based on the load of the system
- ❖ **Algorithm phases:**
 1. **build:** **hashes R** on the **join attribute**, **sends** it to the **target p nodes** that build a hash table for the incoming tuples
 2. **probe:** **sends S associatively** to the **target p nodes** that probe the hash table for each incoming tuple
 3. join each p node and merge all

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



42

42

Parallel Processing - Costs

- ❖ Join processing is achieved with a degree of parallelism of **n processors/nodes**
 - But, each algorithm requires moving at least one of the operand relations (R or S)
- ❖ **Ideal scenario**: no skew in the, and no overhead due to the parallel evaluation
 - **expected speed-up: 1/n**
- ❖ But considering skew and overheads, the time taken by a parallel operation can be estimated as:
 - $T_{cost} = T_{part} + T_{asm} + \max(T_0, T_1, \dots, T_{n-1})$
 - T_{part} - time for partitioning the relations (including communications costs)
 - T_{asm} - time for assembling the results
 - T_i - time taken for the operation at processor P_i . This needs to be estimated taking into account the skew, and the time wasted in contentions



43

43

Parallel Query Optimization

- ❖ The **objective** is to **select the “best” parallel execution plan** for a query using the following components:
 - Search space
 - Alternative model 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
- ❖ **Target**: **minimize the movement of data** among machines



44

44

Load Balancing

- ❖ **Balancing** the **load** of **different transactions** and **queries among different nodes** is essential to **maximize throughput**
- ❖ **Problems** arise for **intra-operator parallelism** with **skewed data distributions**
 - attribute data skew (AVS)
 - inherent to dataset (e.g., there are more citizens in Paris than in Aveiro).
 - tuple placement skew (TPS)
 - introduced when the data are initially partitioned (e.g., with range partitioning)
 - selectivity skew (SS)
 - introduced when there is variation in the selectivity of select predicates on each node
 - redistribution skew (RS)
 - occurs in the redistribution step between two operators (similar to TPS)
 - join product skew (JPS)
 - occurs because the join selectivity may vary between nodes
- ❖ **Solutions**
 - sophisticated parallel algorithms that deal with skew
 - dynamic processor allocation (at execution time)

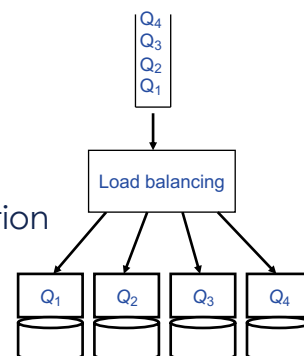


45

45

Load Balancing in a DB Cluster

- ❖ Choose the node to execute Q_i
 - round robin
 - the least loaded
 - need to get load information
- ❖ Failover
 - If the N node fails, N's queries are assumed by other node
- ❖ In case of interference/contention
 - data of an overloaded node are replicated to another node

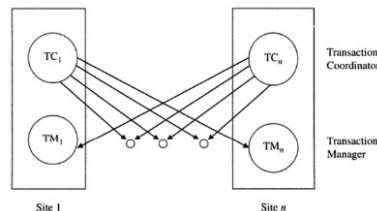


46

46

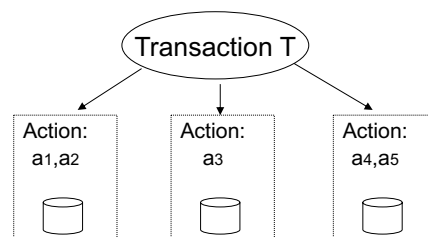
Distributed Transactions

- ❖ Transaction may access data at several sites
- ❖ Each site has a **transaction coordinator** for:
 - **starting** the execution of every transactions at that site
 - **Breaking transaction and distributing** sub-transactions
 - coordinating the **termination** of each transaction
 - may result in the transaction being committed or aborted at all sites
- ❖ Each site has a **transaction manager** responsible for:
 - maintaining a log for recovery purposes
 - coordinating the concurrent execution of the transactions executing at that site



Distributed commit problem

- ❖ **Commit** must be **atomic**...
- ❖ How a distributed transaction that has components at several sites can execute atomically?



- ❖ **Solution:** Two-phase commit (2PC), Centralized 2PC, Distributed 2PC, Linear 2PC, etc.

Example: Two-phase commit protocol

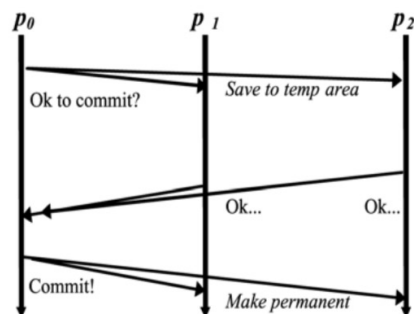
- ❖ **First phase - coordinator collecting a vote** (commit or abort) **from each participant**
 - Participant stores partial results in permanent storage before voting
- ❖ **Second phase - coordinator makes a decision**
 - **if** all participants want to commit and no one has crashed, coordinator multicasts “commit” message
 - everyone commits
 - if participant fails, then on recovery, can get commit msg from coordinator
 - **else** if any participant has crashed or aborted, coordinator multicasts “abort” message to all participants
 - everyone aborts



49

49

Two-phase commit protocol



Coordinator :

- multicast: *ok to commit?*
- collect replies
- all *ok* => *send commit*
- else => *send abort*

Participant:

- ok to commit* =>
- save to temp area, reply *ok*
- commit* =>
- make change permanent
- abort* =>
- delete temp area



50

50

Summary

- ❖ Centralized vs Distributed vs Parallelized Systems
- ❖ Parallel Databases
 - Concept / Objectives
 - Architectures
 - Types of Parallelism
 - DBMS Techniques
 - Data Placement
 - Processing Algorithms
 - Query Optimization
 - Transaction Management



51

51

Resources

- ❖ Martin Kleppmann, ***Designing Data-Intensive Applications***, O'Reilly Media, Inc., 2017.
- ❖ M. Tamer Ozsu, Patrick Valduriez, ***Principles Of Distributed Database Systems*** – 3rd ed, Springer, 2011.
- ❖ Abraham Silberschatz, Henry F. Korth, S. Sudarshan, ***Database System Concepts*** – 6th ed, McGraw-Hill, 2010.



52

52