PARALLEL DATABASE TECHNOLOGY

Kien A. Hua

School of Computer Science University of Central Florida Orlando, FL 32816-2362

Topics Results Queries Parallelizing query QUERY OPTIMIZATION optimization (4)techniques **EXECUTOR** Parallel algorithms (3) Data placement **STORAGE MANAGER** techniques (2)HARDWARE Parallel architectures (1)

Transaction Processing (5)



- A database structure is a collection of *tables*.
- Each table is organized into rows and columns.
- The persistent objects of an application are captured in these tables.

Relational Operator: **SCAN**

```
SELECT NAME
FROM EMPLOYEE
WHERE ADDR = "Orlando";
```



Relational Operator: **JOIN**



Hash-Based Join



"Divide and Concur" done sequentially



"Divide and Concur" Done on Four Processing Nodes

The same divide and concur can be done on four processing nodes in parallel.

Each PN must finish its read and write for the current phase before the next phase can start



Bucket Sizes and I/O Costs



Bucket B does not fit in the memory in its entirety. Bucket A must be loaded several times.

Bucket B(1) fits in the memory. A(1) needs to be loaded only once.

Speedup and Scaleup

The ideal parallel systems demonstrates two key properties:

1. Linear Speedup:

$$Speedup = \frac{small_system_elapsed_time}{big_system_elapsed_time}$$

Linear Speedup : Twice as much hardware can perform the task in half the elapse time (i.e., speedup = number of processors.)

2. Linear Scaleup:

 $Scaleup = \frac{small_system_elapsed_time_on_small_problem}{big_system_elapsed_time_on_big_problem}$

Linear Scaleup : Twice as much hardware can perform twice as large a task in the same elapsed time (i.e., scaleup = 1.)

Barriers to Parallelism

• Startup:

The time needed to start a parallel operation (thread creation/connection overhead) 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.

- The ideal database machine has:
 - 1. a single infinitely fast processor,
 - 2. an infinitely large memory with infinite bandwidth

\longrightarrow Unfortunately, technology is not delivering such machines !

- The challenge is:
 - 1. to build an infinitely fast processor out of infinitely many processors of finite speed, and
 - 2. to build an infinitely large memory with infinitely many storage units of finite speed.

Performance of Hardware Components

• <u>Processor</u>:

- Density increases by 25% per year.
- Speed doubles in three years.

• <u>Memory</u>:

- Density increases by 60% per year.
- Cycle time decreases by $\frac{1}{3}$ in ten years.

• <u>Disk</u>:

- Density increases by 25% per year.
- Cycle time decreases by $\frac{1}{3}$ in ten years.

The Database Problem: The I/O bottleneck will worsen.

Hardware Architectures



Shared Nothing is more scalable for very large database systems





Shared Disk Architecture 4K page **Communication Network** Memory Memory Memory Memory Update D D D

Cross Interrogation for a small change to a page. Processing units interfere each other even they work on different records of the same page.

Hybrid Architecture



- SE clusters are interconnected through a communication network to form an SN structure at the inter-cluster level.
- This approach minimizes the communication overhead associated with the SN structure, and yet each cluster size is kept small within the limitation of the local memory and I/O bandwidth.
- Examples of this architecture include Sequent computers, NCR 5100M and Bull PowerCluster.
- Some of the DBMSs designed for this structure are the Teradata Database System for the NCR WorldMark 5100 computer, Sybase MPP, Informix Online Extended Parallel Server.

Parallelism in Relational Data Model

• **<u>Pipeline Parallelism</u>**: If one operator sends its output to another, the two operators can execute in parallel.



• **<u>Partitioned Parallelism</u>**: By taking the large relational operators and partitioning their inputs and outputs, it is possible to turn one big job into many concurrent independent little ones.



Merge & Split Operators



- Split operator is used to partition or replicate the stream of tuples
- With split and merge operators, a web of simple sequential dataflow nodes can be connected to form a parallel execution plan

Data Partitioning Strategies

Data partitioning is the key to partitioned execution:

- **Round-Robin**: maps the ith tuple to disk $i \mod n$.
- Hash Partitioning: maps each tuple to a disk location based on a hash function.
- **Range Partitioning**: maps contiguous attribute ranges of a relation to various disks.



18

Comparing Data Partitioning Strategies

• Round Robin Partitioning:

Advantage: simple

Disadvantage: It does not support associative search.

• Hash Partitioning:

Advantage:	Associative access to the tuples with a
	specific attribute value can be directed
	to a single disk.

Disadvantage: It tends to randomize data rather than cluster it.

• Range Partitioning:

Advantage:	It is good	for	associative	search	and
	clustering	date	<i>a</i> .		

Disadvantage: It risks execution skew in which all the execution occurs in one partition.

Horizontal Data Partitioning



Query 1:Retrieve the names of students who havea \mathbf{GPA} better than 2.0.

- \implies Only P2 and P3 can participate.
- **Query 2**: Retrieve the names of students who **major** in Anthropology.
 - \implies The whole file must be searched.

Multidimensional Data Partitioning



<u>Advantages</u>:

- Degree of parallelism is maximized (i.e., using as many processing nodes as possible)
- Search space is minimized (i.e., searching only relevant data blocks)

Query Shape:	The shape of the data sub- space accessed by a range query.
Square Query:	The query shape is a square.
Row Query:	The query shape is a rect- angle containing a number of rows.
Column Query:	The query shape is a rect- angle containing a number

of columns.

Optimality

- A data allocation strategy is **usage optimal** with respect to a query type if the execution of these queries can always use all the PNs available in the system.
- A data allocation strategy is **balance optimal** with respect to a query type if the execution of these queries always results in a balance workload for all the PNs involved.
- A data allocation strategy is **optimal** with respect to a query type if it is usage optimal and balance optimal with respect to this query type.

Coordinate Modulo Declustering (CMD)

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	0
2	3	4	5	6	7	0	1
3	4	5	6	7	0	1	2
4	5	6	7	0	1	2	3
5	6	7	0	1	2	3	4
6	7	0	1	2	3	4	5
7	0	1	2	3	4	5	6

Advantages:

Optimal for row and column queries.

Disadvantages: Poor for square queries.

Hilbert Curve Allocation (HCA) Method



Hilbert Curve Allocation (HCA) Method



Navigate the Hilbert curve to label the data cells



There are 8 processing nodes

- Property: A space-filling curve that preserves locality fairly well
 - ⇒ Two data points which are close to each other in 1D space are also close to each other in the high-dimensional space
- Advantage: Good for square range queries
- Disadvantage: Poor for row and column queries

General Multidimensional Data Allocation (GMDA)

Row	0	0	1	2	3	4	5	6	7	8	< Check row
Row	1	3	4	5	6	7	8	0	1	2	
Row	2	6	7	8	0	1	2	3	4	5	
Row	3	1	2	3	4	5	6	7	8	0	< Check row
Row	4	4	5	6	7	8	0	1	2	3	
Row	5	7	8	0	1	2	3	4	5	6	
Row	6	2	3	4	5	6	7	8	0	1	< Check row
Row	7	5	6	7	8	0	1	2	3	4	
Row	8	8	0	1	2	3	4	5	6	7	

 ${\cal N}$ is the number of processing nodes

<u>Regular Rows</u>: Circular left shift $\lfloor \sqrt{N} \rfloor$ positions.

<u>Check Rows</u>: Circular left shift $\lfloor \sqrt{N} \rfloor + 1$ positions.

<u>Advantages</u>: optimal for row, column, and small square range queries $(|Q| < \lfloor \sqrt{N} \rfloor^2)$.

Handling 3D

A cube with N³ grid blocks can be seen as N 2D planes stacked up in the third dimension



Algorithm 3D_GMDA:

1. We compute the mapping for the first rows of all the 2dimensional planes

by considering these rows collectively as forming a plane along the third dimension. We apply the 2D_GMDA algorithm to this plane, except that the shifting distance is set to $\lfloor \sqrt[3]{N} \rfloor^2$.

2. We apply the 2D_GMDA algorithm to each of the 2-dimensional planes using the shifting distance $\lfloor \sqrt[3]{N} \rfloor$ and the first row already computed in Step 1.

Shifting Distance

- Number of dimension N=3
- Step 1: Shifting distance is



• Step 2: Shifting distance is



Handling Higher Dimensions: Mapping Function

A grid block $(X_1, X_2, ..., X_d)$ is assigned to PN $GeMDA(X_1, X_2, ..., X_d)$, where

$$GeMDA(X_1, ..., X_d) = \left[\sum_{i=2}^d \left\lfloor \frac{X_i \cdot GCD_i}{N} \right\rfloor + \sum_{i=1}^d \left(X_i \cdot Shf_dist_i\right)\right] \mod N,$$

N = number of PNs,

$$Shf_dist_i = \lfloor \sqrt[d]{N} \rfloor^{i-1}$$
, and

$$GCD_i = gcd(Shf_dist_i, N)$$

Optimality Comparison

Allocation	Optimal with respect to									
scheme	row queries	column queries	small square queries							
HCAM	No	No	No							
CMD	Yes	Yes	No							
GeMDA	Yes	Yes	Yes							

Conventional Parallel Hashbased Join: Grace Algorithm



Grace Algorithm

- Hash Phase: The hash buckets are evenly assigned among the PNs. Each PN hashes its local partition of each relation and sends each tuple to the target hash bucket according to the hash result (i.e., the remainder)
- Bucket Tuning Phase: The small buckets are logically combined (disk access not required) to form a larger bucket to better fit the local memory⁺. The buckets of the two relations must be combined in the same way according to the same hash codes (i.e., same remainders).
- Join Phase: The PNs perform in parallel their local joins.

⁺Only one bucket of each bucket pair needs to fit in the memory in its entirety so that the other bucket needs to be scanned only once.

Computation Cost Estimation



If each of the R partitions has 50 pages and each of the S partitions has 100 pages,

- the read cost of the Hash Phase is 50 + 100 = 150 IOs,
- the write cost of the Hash Phase is 50 + 100 = 150 IOs, and
- the total computation cost of the Hash Phase is 150 + 150 = 300 IOs
The Effect of Imbalanced Workloads



Partition Tuning: Largest Processing Time (LPT) First Strategy - Bin Packing



Naive Load Balancing Parallel Hash Join (NBJ)



Partition Tuning: Re-distribute the hash bucket among the PNs using Bin Packing to balance their workload

Kien A. Hua

Adaptive Load Balancing Parallel Hash Join (ABJ)



Workload is balanced among the PN's throughout the computation

What if the partitioning is skew initially ?

ABJ Algorithm

- Hash: Each PN maintains a subbucket for each of the hash buckets. Each PN hashes its local partition of each relation and stores each tuple to the appropriate subbucket according to the hash result (i.e., no data transmission)
- Partition Tuning: The coordinating PN computes the total size of each bucket by summing up the sizes of its subbuckets. This size information is used to run the bin packing algorithm to assign the bucket pairs among the PNs to achieve load balancing.
- Bucket Tuning: The small buckets are logically combined (disk access not required) to form a larger bucket to better fit the local memory⁺. The buckets of the two relations must be combined in the same way according to the same hash codes (i.e., same remainders).
- Join: The PNs perform in parallel their local joins.

Tuple Interleaving Parallel Hash Join (TIJ)



Workload is balanced among the PN's throughout the computation



Sampling-based Load Balancing (SLB) Join Algorithm

- Sampling Phase: Each PN loads a small percentage of its tuples into memory, and hash them into a large number of inmemory hash buckets (hash on the join attribute)
- Partition Tuning: The coordinating PN applies "bin packing" to the in-memory buckets to determine the optimal *bucket* allocation scheme (BAS)
- Split Phase:
 - The in-memory buckets are collected to their destined PN in accordance with the BAS to form the initial partial buckets
 - Each PN loads the remaining tuples and forwards them to their destined hash buckets ("bin packing" not needed)



• Join Phase: Each PN performs the local joins of respectively matching buckets

nCUBE/2 Results: <u>SLB vs. ABJ vs. GRACE</u>



- The performance of SLB approaches that of GRACE on very mild skew conditions, and
- it can avoid the disastrous performance that GRACE suffers on severe skew conditions.

Pipelining Hash-Join Algorithms

- <u>Two-Phase Approach</u>: Uutput stream Hash Table First operand Second operand
 - -<u>Advantage</u>: requires only one hash table.
 - <u>Disadvantage</u>: pipelining along the outer relation must be suspended during the build phase (i.e., building the hash table).
- <u>One-Phase Approach</u>: As a tuple comes in, it is first inserted into its hash table, and then used to probe that part of the hash table of the other operand that has already been constructed.



- -<u>Advantage</u>: pipelining along both operands is possible.
- <u>Disadvantage</u>: requires larger memery space.

• An SQL aggregate function is a function that operates on groups of tuples.

Example:	SELECT	department, COUNT(*)
	FROM	Employee
	WHERE	age > 50
	GROUP BY	department

• The number of result tuples depends on the selectivity of the GROUP BY attributes (i.e., *department*).

Centralized Merging



Distributed Merging



Repartitioning



Repartitioning



Performance Characteristics

• <u>Centralized Merging Algorithm</u>:

Advantage: It works well when the number of tuples is small.

Disadvantage: The merging phase is sequential.

- Distributed Merging Algorithm:
 - Advantage: The merging step is not a bottleneck.
 - <u>Disadvantage</u>: Since a group value is being accumulted on potentially all the PNs the overall memory requirement can be large.
- <u>Repartitioning Algorithm</u>:
 - <u>Advantage</u>: It reduces the memory requrement as each group value is stored in one place only.

Disadvantage: It incurs more network traffic.

Coventional Aggregation Algorithms

• Centralized Merging (CM) Algorithm:

Phase 1: Each PN does aggregation on its local tuples.

Phase 2: The local aggregate values are merged at a predetermined central coordinator.

• Distributed Merging (DM) Algorithm:

Phase 1: Each PN does aggregation on its local tuples.

- **Phase 2:** The local aggregate values are then hash-partitioned (based on the GROUP BY attribute) and the PNs merge these local aggregate values in parallel.
- Repartitioning (Rep) Algorithm:
 - **Phase 1:** The relation is repartitioned using the GROUP BY attributes.
 - **Phase 2:** The PNs do aggreation on their local partitions in parallel.

Performance Comparison:

- CM and DM work well when the number of result tuples is small.
- Rep works better when the number of groups is large.

Adaptive Aggregation Algorithms

• <u>Sampling Based (Samp) Approach:</u>

This step incurs overhead

- CM algorithm is first applied to a small Page-oriented random sample of the relation.
- If the number of groups obtained from the sample is small then DM strategy is used; Rep algorithm is used otherwise.
- Adaptive DM (A-DM) Algorithm:



- This algorithm starts with the DM strategy under the common case assumption that the number of group is small.
- However, if the algorithm detects that the number of groups is large (i.e., memory full is detected) it switches to the Rep strategy.
- The Adaptive Repartitioning (A-Rep) Algorithm:
 - This algorithm starts with the Rep strategy.
 - It switches to DM if the number of groups is not large enough (i.e., number of groups is too few given the number of seen tuples).

Performance Comparison:

- In general, A-DM performs the best.
- However, A-Rep should be used if the number of groups is suspected to be very large.

Implementation Techniques for A-DM

• <u>Global Switch</u>:

- When the first PN detects a memory full condition, it informs all the PNs to switch to the Rep strategy.
- Each PN first partitions and sends the so far accumulated local results to PNs they hash to. Then, it proceeds to read and repartition the remaining tuples.
- Once the repartitioning phase is complete, the PNs do aggregate on the local partitions in parallel (as in the Rep algorithm).

• <u>Local Switch</u>:

- A PN upon detecting memory full stops processing its local tuples. It first partitions and sends the so far accumulated local results to the PNs they hash to. Then it proceeds to read and repartition the remaining tuples.
- During Phase one, one set of PNs may be executing the DM algorithm while other are executing the Rep algorithm. When the latter receives an aggregate value from another PN, it accumulates it to the corresponding local aggregate value.
- Once all PNs have completed their Phase 1, The local aggregate values are merged as in the DM algorithm.

A-DM: Global Switch



Switching to Rep

PN2 and PN3 must also switch with PN1 (i.e., global switch)

- Step 1: Prepare for Repartitioning
- Step 2: Apply Repartitioning to the remaining tuples



SQL (Structured Query Language)



```
An SQL query:SELECTENAMEFROMEMPLOYEE, WORKSON, PROJECTWHEREPNAME= 'database' ANDPNUM = PNO ANDENO = ENUM ANDBDATE > '1965'
```

- SQL is *nonprocedural*.
- The Compiler must generate the *execution plan*.
 - 1. Transforms the query from SQL into *relational algebra*.
 - 2. Restructures (optimizes) the algebra to improve performance.

Relational Algebra

Relation T1				Relation T2		
ENAME	SALARY	ENUM		ENUM	ADDRESS	BDATE
Andrew	\$98,000	005	<	001	Orlando	1964
Casey	\$150,000	003	<	003	New York	1966
James	\$120,000	007		005	Los Angeles	1968
Kathleen	\$115,00	001		007	London	1958

• **<u>Select</u>**: Selects rows.

$$\sigma_{SALARY \ge 120,000}(T1) = \begin{cases} (Casey, 150000, 003) \\ (James, 120000, 007) \end{cases}$$

• **<u>Project</u>**: Selects columns.

$$\pi_{ENAME,SALARY}(T1) = \begin{cases} (Andrew, 98000), \\ (Casey, 150000), \\ (James, 120000), \\ (Kathleen, 115000) \end{cases}$$

• **Cartesian Product**: Selects all possible combinations.

 $T1 \times T2 = \begin{cases} (Andrew, 98000, 005, 001, Orlando, 1964), \\ (Andrew, 98000, 005, 003, New York, 1966), \\ \vdots \\ (Kathleen, 150000, 001, 005, Los Angeles, 1968) \\ (Kathleen, 115000, 001, 007, London, 1958) \end{cases}$

• Join: Selects some combinations.

$$T1 \bowtie T2 = \begin{cases} (Andrew, 98000, 005, Los Angeles, 1968), \\ (Casey, 150000, 003, New York, 1966), \\ (James, 120000, 007, London, 1958), \\ (Kathleen, 115000, 001, Orlando, 1964) \end{cases}$$

Transforming SQL into Algebra





This query tree (procedure) will compute the correct result. However, the performance will be very poor. \implies needs optimization !

<u>GOAL</u>: Reducing the sizes of the intermediate results as quickly as possible.

STRATEGY:

- 1. Move SELECTs and PROJECTs as far down the query tree as possible.
- 2. Among SELECTs, reordering the tree to perform the one with lowest selectivity factor first.
- 3. Among JOINs, reordering the tree to perform the one with lowest join selectivity first.

Example: Apply SELECTs First



Example: Replace " $\sigma - \times$ " by " \bowtie "

Before Optimization



Example: Move PROJECTs Down

Before Optimization



After Optimization



Parallelizing Query Optimizer

Relations are fragmented and allocated to multiple processing nodes:

- The role of a parallelizing optimizer is to map a query on a global relations into a sequence of local operations acting on local relation fragments
- Besides the choice of ordering relational operations, the parallelizing optimizer must select the best PNs to process data



Parallelizing Query Optimization



Parallelizing Optimizer:

- Parallelizes the relational operators.
- Selects the best processing nodes for each parallelized relational operator.

Parallelizing Example





(5) Select the best processing node for each fragment operator

Parallelizing Query Optimization

- 1. Determines which fragments are involved and transforms the global operators into fragment operators.
- 2. Eliminates useless fragment operators.
- 3. Finds the "best" ordering of the fragment operators.
- 4. Selects the best processing node for each fragment operator and specifies the communication operations.



Prototype at UCF

- A prototype of a shared-nothing system is implanted on a 64processor nCUBE/2 computer
- Our system was implemented to demonstrate:
 - GeMDA multidimensional data partitioning technique,
 - Dynamic optimization scheme with load balancing capability, and
 - Competition-based scheduling policy



Software Componets

- Storage Manager: This component manages physical disk devices and schedules all I/O activities. It provides a sequential scan interface to the query processing facilities.
- **Catalog Manager**: It acts as a central repository of all global and fragment schemas.
- Load Utility: This program allows the users to populate a relation using an external file. It distributes the fragments of a relation across the processing nodes using GeMDA.
- Query Translator: This component provides an interface for queries. It translates an SQL query into a query graph. It also caches the global schema information locally.
- Query Executor: This component performs dynamic query optimization. It schedules the execution of the operators in the query graph.
- **Operator Routines**: Each routine implements a primitive database operator. To execute an operator in a query graph, the Query Executor calls the appropriate operator routine to carry out the underlying operation.
- **Presentation Manager**: This module provides an interactive interface for the user to create/destroy tables, and query the database. The user can also use this interface to browse query results.



Windows


Server Class

JOIN

1011



Server Class: A group of processes, each providing the same service (e.g., JOIN).

Windows

An operator server

Many Server Classes per PN



Parallel Processing





Parallelized Execution Plan **Query Executor** INSERT INSERT INSERT assigns operators in the parallelized execution plan to JOIN logical servers in the different server pools of the different PNs SCAN for parallel execution **SCAN** Assign operators to logical servers JOIN JOIN JOIN **SCAN** SCAN INSERT INSERT **INSER** Server pool Server pool Server pool Executor A logical A parallel server execution plan PN1 PN2 PN3

A logical server is software running in a process, capable of performing a certain basic database operation (e.g., INSERT)



Advantage: Fair.Disadvantage: System utilization is not maximized.

Each operator server is associated with a processing node

Competition-based: Potential drawbacks



wait for Query 2 to leave the FIFO waiting queue; ...

Competition-based: Potential drawbacks



wait for Query 2 to leave the FIFO waiting queue; ...

With some planning (vs. FCFS), the following schedule achieves better system utilization:



Planning-based Scheduling



- Scheduler: It plans and schedules the execution of operators from multiple queries currently within the scheduling window
- **Coordinator**: It coordinates the parallel execution of each query operator scheduled by the Scheduler

Planning-based Scheduling



- Scheduler: It plans and schedules the execution of operators from multiple queries currently within the scheduling window
- **Coordinator**: It coordinates the parallel execution of each query operator scheduled by the Scheduler

Planning-based Scheduling



- Advantage: Better system utilization
- Disadvantage: Less fair

Hardware Organization



IFP: Interface Processor ACP: Access Processor

- Catalog Manager, Query Manager, and Scheduler processes run on IFP's
- Operator processes run on ACP's for parallel query computation

Structure of Operator Processes



- The output is demultiplexed through a split table.
- When the process detects the end of its input stream,
 - $-\operatorname{it}$ first closes the output streams and
 - then sends a control message to its coordinator process indicating that it has completed execution.

Example: Operator and Process Structure

• Query Tree:





Transaction Processing

The consistency and reliability aspects of transactions are due to four properties

- Atomicity: A transaction is either performed in its entirety or not performed at all
- Consistency: A correct execution of the transaction must take the database from one consistent state to another
- Isolation: A transaction should not make its updates visible to other transaction until it is committed
- **Durability**: Once a transaction changes the database and changes are committed, these changes must never be lost because of subsequent failure



Transaction Manager



• Lock Manager:

- Each local lock manager is responsible for the lock units local to that processing node.
- They provide *concurrency control*.

Isolation property

• Log Manager:

- Each local log manager logs the local database operations.
- They provide *recovery services*.



- Any *schedule* generated by a concurrency control algorithm that obeys the 2PL protocol is serializable (i.e., the isolation property is guaranteed).
- 2PL is difficult to implement. The lock manager has to know:
 - 1. the transaction has obtained all its locks, and
 - 2. the transaction no longer needs to access the data item in question. (so that the lock can be released).
- Cascading aborts can occur.
 (because transactions reveal updates before they commit)

Strict Two-Phase Locking Protocol



The lock manager releases all the locks together when the transaction terminates (commits or aborts).

Wait-for Graph

- If a transaction reads an object, the transaction depends on that object version
- If the transaction writes an object, the resulting object version depends on the writing transaction.





 $\begin{array}{c} \text{READ} \rightarrow \text{WRITE} \\ \text{dependency} \end{array}$

WRITE \rightarrow READ dependency

WRITE \rightarrow WRITE dependency

Handling Deadlocks

• <u>Detection and Resolution</u>:

- Abort and restart a transaction if it has waited for a lock for a long time.
- Detect cycles in the wait-for graph and select a transaction (involved in a cycle) to abort.

• <u>Prevention</u>:

If T_i requires a lock held by T_j ,

- \star If T_i is older $\implies T_i$ can wait.
- $\star \text{ If } T_i \text{ is younger } \Longrightarrow T_i \text{ is aborted and restarted} \\ \text{ with the same timestamp.}$



Distributed Deadlock Detection



• When a transaction is blocked, it sends a special probe message to the blocking transaction.

The message consists of three numbers: the transaction that just blocked, the transaction sending the message, and the transaction to whom it is being sent.

- When the message arrives, the recipient checks to see if it itself is waiting for any transaction. If so, the message is updated, replacing the second field by its own TID and the third one by the TID of the transaction it is waiting for. The message is then sent to the blocking transaction.
- If a message goes all the way around and come back to the original sender, a deadlock is detected.

Two-Phase Commit Protocol

To ensure the atomicity property, a 2P commit protocol can be used to coordinate the commit process among subtransactions.



Recovery

- An entry is made in the local log file at a processing node each time one of the following commands is issued by a transaction:
- begin transaction
 write (insert, delete, update)
 commit transaction
 abort transaction
 abort transaction
 Write-ahead log protocol: An entry in the log file
 It is essential that log records be written before the corresponding write to the database.

If log entry wasn't saved before the crash, corresponding change was not applied to database!

 If there is no commit transaction entry in the log for a particular transaction, then that transaction was still active at the time of failure and must therefore be undone.

This is to ensure the atomicity property

Commercial Product: Teradata DBC/1012



IFP: Interface Processor AMP: Access Module Processor COP: Communication Processor

- It may have over 1,000 processors and many thousands of disks.
- Each relation is hash partitioned over a subset of the AMPs.
- Near-linear speedup and scaleup on queries have been demonstrated for systems containing over 100 processors.

Ynet (1)





Teradata DBC/1012: Distribution of Data

- The fallback copy ensures that the data remains available on other AMPs if an AMP should fail.
- In the following example, if AMPs 4 and 7 were to fail simultaneously, however, there would be a loss of data availability.



- Additional data protection can be achieved by "clustering" the AMPs in groups.
- In the following example, If both AMPs 4 and 7 were to fail, all data would still be available.



Commercial Product: Tandem NonStop SQL



- Tandem systems run the applications on the same processors as the database servers.
- Relations may be range partitioned across multiple disks.
- It is primarily designed for OLTP. It scales linearly well beyond the largest reported mainframes on the TPC-A benchmarks.
- It is three times cheaper than a comparable mainframe system.