Transaction Management

- **Atomicity**

  Either all or none of the transaction's operations are performed. Atomicity requires that if a transaction is interrupted by a failure, its partial results are undone.

Reasons for transaction not completed: Transaction aborts or system crashes.

Commitment: completion of a transaction

Transaction primitives*:* BEGIN, COMMIT, ABORT

- Global of transaction management: Efficient, reliable, and concurrent execution of transactions
- Agents: A local process which performs some actions on behalf of an application
- **Root Agent:**

  -Issuing begin transaction, commit, and abort primitives

  -Create mew agent

Failure Recovery

Basic Techniques: LOG

A log contains information for undoing or redoing all actions which are performed by transactions.

Undo: Reconstruct the database as prior to its execution (e.g.. abort)

Redo: Perform again its action (e.g.. failure of volatile storage before writing on to stable storage, but already committed)

Undo and redo must be independent. Performing them several times should be equivalent to performing them once.

A log contains:

1. Transaction ID
2. Record ID
3. Type of action (insert, delete, modify)
4. The old record value (required for undo)
5. The new record value (required for redo)

Failure Recovery (cont'd)

A log contains:

6. Information for recovery (e.g.. a pointer to the previous log record of the same transaction).
7. Transaction status (begin, abort, commit)

Log Write-ahead protocol

- Before performing a database update, log record recorded on stable storage
- Before committing a transaction, all log records of the transaction must have recorded on stable storage.

Recovery Procedure Via Check Points

Check points are operations which are periodically performed (e.g. few minutes) which write the following to stable storage.

- All log records and all database updates which are still in volatile storage.
- Check point record which contains the indication of transactions which are active at the time when check point is done.

Local Transaction Manager (LTM)

Provides transaction management at local site, for example, local begin, local commit, local abort, perform subtransaction (local transaction).

Distributed Transaction Manager (DTM)

Provides global transaction management.

LTM has the capabilities

- Ensuring the atomicity of a subtransaction.
- Write record on stable storage on behave of DTM.

Atomicity at LTM is not sufficient for atomicity at DTM (i.e. single site vs. all sites).

Two Phase Commit Protocol

- *Coordinator:* Making the final commit or abort decision (e.g.. DTM).
- *Participants:* Responsible for local subtransactions (e.g. LTM).
- *Basic Idea:* Unique decisions for all participants with respect to committing or aborting all the local subtransactions.
- *1st Phase:* To reach a common decision
- *2nd Phase:* Global commit, or global abort, (recording the decision on the stable storage.)

Phase 1

- The coordinator asks all the participants to prepare for commitment.
- Each participant answers READY if it is ready to commit and willing to do so. Each participant record on the stable storage.
    1) all information which is required for locally committing the subtransactions.
    2) “ready” log record must be recorded on the stable storage
- The coordinator records a “prepare” log on the stable storage which contains all the participants’ identification, and also activates a time out mechanism.

Phase 2

- The coordinator recording on the stable storage of its decision “global commit” or “global abort”
- The coordinator informs all the participants of its decision
- All participants write a commit or abort record on the log (assure local subtransaction will not be lost).
- All participants send a final acknowledgment message to the coordinator and perform the actions required for committing or aborting the subtransaction.
- Coordinator writes a “complete” record on the stable storage.

Coordinator: Write a “prepare” record in the log:

Send PREPARE message and activate time-out

Participant: Wait for PREPARE message:

If the participant is willing to commit then
begin
Write subtransaction’s records in the log;
Write “ready” record in the log;
Send READY answer message to coordinator
end
else begin
Write “abort” record in the log;
Send ABORT answer message to coordinator

Coordinator: Wait for ANSWER message (READY or ABORT) from all participants or time-out;

If time-out expired or some answer message is ABORT then

begin

Write “global_abort” record the log;

Send ABORT command message to all participants

end

Participant: Wait for command message;

Write “abort” or “commit” record in the log:

Send the ACK message to coordinator;

Execute command

Coordinator: Wait for ACK message form all participants:

Write “complete” record in the log

Basic 2-phase-commitment protocol.

Elimination of the PREPARE Message
2P Commit

Coordinator:

Write prepare record in the log;

Request operations for participants, and activate time-out;

Wait for completion of participants (READY message) or time-out expired:

Write global_commit or global_abort record in the log;

Send command message to all participants.

Participant:

Receive request for operation:

Perform local processing and write log records:

Elimination of the PREPARE Message
2P Commit (Cont'd)

Participant:

Send READY message and write ready record in the log:

Wait for command message:

Write commit or abort records in the log;

Execute command.

THE CONSISTENCY PROBLEM IN A DISTRIBUTED DATABASE SYSTEM

- MULTIPLE COPIES OF THE SAME DATA AT DIFFERENT SITES IMPROVE
    - AVAILABILITY
    - BETTER RESPONSE TIME

- EVERY UPDATE WILL RESULT IN A LOCAL EXECUTION AND A SEQUENCE OF UPDATES SENT TO THE VARIOUS SITES WHERE THERE IS A COPY OF THE DATABASE

CONCURRENCY CONTROL

- PURPOSE: TO GIVE EACH USER THE ILLUSION THAT HE IS EXECUTING ALONE ON A DEDICATED SYSTEM WHEN, IN FACT, MANY USERS ARE EXECUTING SIMULTANEOUSLY ON A SHARED SYSTEM.
- GOALS: MUTUAL CONSISTENCY

  INTERNAL CONSISTENCY
- PROBLEMS: DATA STORED AT MULTIPLE SITES

  COMMUNICATION DELAYS

CONCURRENCY CONTROL IS A COMPONENT OF A DISTRIBUTED DATABASE MANAGEMENT SYSTEM

CRITERIA FOR CONSISTENCY

- MUTUAL CONSISTENCY AMONG THE REDUNDANT COPIES
- INTERNAL CONSISTENCY OF EACH COPY

- ANY ALTERATIONS OF A DATA ITEM MUST BE PERFORMED IN ALL THE COPIES
- TWO ALTERATIONS TO A DATA ITEM MUST BE PERFORMED IN THE SAME ORDER IN ALL COPIES

A GOOD SOLUTION MUST BE

- DEADLOCK FREE
- SPEED INDEPENDENT
- PARTIALLY OPERABLE

CONCURRENCY CONTROL

CORRECTNESS => SERIALIZABLE EXECUTIONS

SERIALIZABLE EXECUTION=>SERIAL EXECUTION

SERIAL EXECUTION=>NO CONCURRENCY

- TWO OPERATIONS ARE SAID TO CONFLICT IF THEY OPERATE ON THE SAME DATA ITEM AND AT LEAST ONE IS A WRITE.
- TWO TYPES OF CONFLICTS:
  - WRITE-WRITE (WW)
  - READ-WRITE (RW)

CONCURRENCY CONTROL (CONT'D)

- BERNSTEIN AND GOODMAN SEPARATE TECHNIQUES MAY BE USED TO INSURE RW AND WW SYNCHRONIZATION.

THE TWO TECHNIQUES CAN BE "GLUED" TOGETHER VIA AN INTERFACE WHICH ASSURES ONE SERIAL ORDER CONSISTENT WITH BOTH.

DEFINITIONS OF CONCURRENCY CONTROL

- A SCHEDULE (HISTORY OR LOG) IS A SEQUENCE OF OPERATIONS PERFORMED BY TRANSACTIONS

  S 1: $R_i\ (X)\ R_j\ (X)\ W_i\ (y)\ R_k\ (y)\ W_j\ (X)$

- TWO TRANSACTIONS T_i AND T_j EXECUTE SERIALLY ON A SCHEDULE S IF THE LAST OPERATION OF T_i PRECEDES THE FIRST OPERATION OF T_j IN S; OTHERWISE THEY EXECUTE CONCURRENTLY IN IT.
- A SCHEDULE IS SERIAL IF NO TRANSACTIONS EXECUTE CONCURRENTLY IN IT.

FOR EXAMPLE

S 2: $R\,i\ (X) Wi\ (X) R_j\ (X) W_j\ (y) R_k\ (y) W_k\ (X) = T_i T_j T_k$

DEFINITIONS OF CONCURRENCY CONTROL (CONT'D)

- GIVEN A SCHEDULE S, OPERATION O_i PRECEDES O_j ($O_i < O_j$), IF O_i APPEARS TO THE LEFT OF O_j IN S.

- A SCHEDULE IS CORRECT IF IT IS SERIALIZABLE; IT IS COMPUTATIONALLY EQUIVALENT TO A SERIAL SCHEDULE.

SERIALIZABILITY IN A DISTRIBUTED DATABASE

- SERIALIZABILITY OF LOCAL SCHEDULES IS NOT SUFFICIENT TO ENSURE THE CORRECTNESS OF THE EXECUTIONS OF A SET OF DISTRIBUTED TRANSACTIONS.

- FOR EXAMPLE:

  S1: $T_i > T_j$

  S2: $T_j > T_i$

- THUS, THE EXECUTION OF T_i..., T_n IS CORRECT IF

  1)EACH LOCAL SCHEDULE S_k IS SERIALIZABLE

SERIALIZABILITY IN A DISTRIBUTED DATABASE (CONT'D)

- THUS, THE EXECUTION OF T_i..., T_n IS CORRECT IF

  2) THERE EXISTS A TOTAL ORDER OF T_1, ..., T_n SUCH THAT IF $T_i < T_j$ IN THIS TOTAL ORDERING, THEN THERE IS A SERIAL SCHEDULE $S_{k'}$, SUCH THAT S_k IS EQUIVALENT TO $S_{k'}$ AND $T_i < T_j$ IN $S_{k'}$ FOR SITE K.

CONSISTENCY CONTROL TECHNIQUES

- TIME STAMPS
- LOCKING
  - PRIMARY SITE LOCKING
- EXCLUSIVE-WRITER
  - EXCLUSIVE-WRITER USING SEQUENCE NUMBER
  - EXCLUSIVE-WRITER USING SEQUENCE NUMBER WITH LOCK OPTIONS

TWO PHASE LOCKING (2PL)

- READ AND WRITE LOCKS
- LOCKS MAY BE OBTAINED ONLY IF THEY DO NOT CONFLICT WITH A LOCK OWNED BY ANOTHER TRANSACTIONS.
- CONFLICTS OCCUR ONLY IF THE LOCKS REFER TO THE SAME DATA ITEM AND:
    - RW - ONE IS A READ LOCK AND THE OTHER IS A WRITE LOCK
    - WW - BOTH ARE WRITE LOCKS

TWO PHASE LOCKING (2PL) (CONT'D)

"TWO-PHASED-NESS"

- GROWING PHASE
- LOCKED-POINT
- SHRINKING PHASE
- ONCE A LOCK IS RELEASED, NO NEW LOCKS MAY BE OBTAINED
- LOCKED POINTS DETERMINES SERIALIZATION ORDER

CENTRALIZED LOCKING ALGORITHM

- ALL REQUESTS FOR RESOURCES MUST BE SENT TO ONE SITE CALLED THE LOCK CONTROLLER
- THE LOCK CONTROLLER MAINTAINS A LOCK TABLE FOR ALL THE RESOURCES IN THE SYSTEM
- BEFORE A SITE CAN EXECUTE A TRANSACTION, IT MUST OBTAIN THE REQUIRED LOCKS FROM THE LOCK CONTROLLER
- ADVANTAGE:
  - FEWER MESSAGES REQUIRED FOR SETTING LOCKS THAN IN DISTRIBUTED LOCKING ALGORITHMS

CENTRALIZED LOCKING ALGORITHM (CONT'D)

- DISADVANTAGES:
  - POOR RELIABILITY; BACKUP SYSTEM REQUIRED
  - LOCK CONTROLLER MUST HANDLE LARGE TRAFFIC VOLUME

DISTRIBUTED LOCKING ALGORITHM

1. WAIT FOR A TRANSACTION REQUEST FROM USER
2. SEND n LOCK REQUEST MESSAGE
3. IN CASE OF ANY LOCK REJECT SEND LOCK RELEASE AND GO TO 2 TO RETRY AFTER A RANDOM INTERNAL OF TIME
4. PERFORM LOCAL TRANSACTION AND SEND n UPDATE MESSAGES
5. WAIT FOR UPDATE ACK MESSAGES

DISTRIBUTED LOCKING ALGORITHM (CONT'D)

6. SEND n LOCK RELEASES, NOTIFY USER THE TRANSACTION IS DONE, GO TO 1

- 5 n COMPUTER TO COMPUTER MESSAGE
- TIME CONSUMING
- LONG DELAY

VOTING ALGORITHM

- THE DATA BASE MANAGER PROCESS SENDS AN UPDATE REQUEST TO THE OTHER DBMP

- THE REQUESTS CONTAIN THE VARIABLES THAT PARTICIPATE IN THE QUERY WITH THEIR TIME STAMPS, AND THE NEW VALUES FOR THE UPDATES VARIABLES.

- EACH DBMP VOTES OK, REJ, OR PASS, OR DEFERS VOTING

- THE TRANSACTION IS ACCEPTED OF THE MAJORITY VOTED OK

IF TWO REQUESTS ARE ACCEPTED, IT MEANS THAT AT LEAST ONE DBMP VOTED OK FOR BOTH.

BROADCAST	2.5 n TRANSMISSION
DAISY CHAIN	1.5 n TRANSMISSION

CHARACTERISTICS OF PRIMARY SITE LOCKING

- SERIALIZABILITY
- MUTUAL CONSISTENCY
- MODERATE TO HIGH COMPLEXITY
- CAN CAUSE DEADLOCKS
- INTERCOMPUTER SYNCHRONIZATION DELAYS

VARIABLE LEVEL OF SYNCHRONIZATION

- GLOBAL DATABASE LOCK IS NOT REQUIRED BY MOST OF THE TRANSACTIONS
- DIFFERENT TYPES OF TRANSACTIONS NEED DIFFERENT LEVELS OF SYNCHRONIZATION
- THE LEVEL OF SYNCHRONIZATION CAN BE REPRESENTED BY ALGORITHMS (PROTOCOLS) WHICH ARE EXECUTED WHEN A TRANSACTION IS REQUESTED.

GOAL: EACH TRANSACTION SHOULD RUN UNDER THE PROTOCOL THAT GIVES THE LEAST DELAY WITHOUT COMPROMISING THE CONSISTENCY.

IN SDD-1, FOUR PROTOCOLS ARE AVAILABLE. DIFFERENT LEVELS OF SYNCHRONIZATION YIELD DIFFERENT DELAYS.

DISADVANTAGE: HIGH OVERHEAD COSTS.

THE EXCLUSIVE-WRITER PROTOCOL

- IMPLEMENTATION REQUIREMENTS
  - EACH COPY OF A FILE HAS AN UPDATE SEQUENCE NUMBER (SN)
- OPERATION
  - ONLY THE EXCLUSIVE-WRITER (EW) DISTRIBUTES FILE UPDATES
  - UPDATING TASKS SENDS TO THE EW'S SITE UPDATE-REQUEST MESSAGES (UPDATE AND SN)
  - BEFORE THE EW DISTRIBUTES A FILE UPDATE, IT INCREMENTS THE FILE'S SN
  - THE EW DETECTS A DATA CONFLICT WHEN THE SN IN THE UPDATE-REQUEST IS LESS THAN THE SN OF THE CORRESPONDING FILE AT THE EW'S SITE

COMPARISON OF PSL AND THE EWP

- PSL
  - NO DISCARDED UPDATES
  - INTERCOMPUTER SYNCHRONIZATION DELAYS
  - CAN CAUSE DEADLOCKS
- EWP
  - CONFLICTING UPDATES ARE DISCARDED (EWP WITHOUT LOCK OPTION)
  - NONITERCOMPUTER SYNCHRONIZATION DELAYS
  - LOWER MESSAGE VOLUME THAN PSL
- DESIGN ISSUE
  - SELECTION OF PRIMARY SITE AND EXCLUSIVE-WRITER SITE
  - LIMITED REPLICATION OF SHARED FILES

COMPARISON OF PSL AND THE EWP

- PERFORMANCE ANALYSIS
  - VOLUME OF MESSAGE
  - RESPONSE TIME
  - PROCESSING OVERHEAD

Escrow Locking

Pat O'Neil

For updating numerical values

- money
- disk space

Uses Primary Site

Lock in advance only the required amount (Errors)

Release excess resources after execution

Advantage:

- Less lock conflict, therefore more data availability
- Less concurrency control overhead for update, good for long transactions

Disadvantage:

- Weak data consistency
- Data usually are inconsistent, but within a certain bond

Escrow Locking (cont'd)

EXAMPLE:

Bank account with $50

Need to clear a check for up to $30
Escrow lock $30 - other $20 is still available
If check is only for $25, return remaining $5

Escrow Locking under Partitioning

Similar to PSL

only Primary Site partition can update

Primary Site may be isolated in a small partition

Further

escrows may be outstanding when partitioning occurs

Solution

- grant the escrow amount to each partition
- based on user profile/history
- based on size/importance of the partitions

Escrow Locking under Partitioning (cont'd)

EXAMPLE:

- Escrow amount = total amount/# of partitions

  Bank account with $50

  If two partitions occur

  escrow $25 in each partition (for that partition to use)

  If some updates require $30, then update will be blocked

- Based on historical information, give different escrow portions to different partitions

  Eg. Escrow for partition A= 35

  Escrow for partition B=15

- Use normal escrow locking in each partition
- reconcile database afterwards

DEADLOCK AVOIDANCE

- Priority
- Timestamps: A transaction's timestamp is the time at which it begins execution. Old transactions have higher priority than younger ones.

Timestamp deadlock prevention schemes

- Assume older transaction has higher priority than younger transaction.

- *Wait-Die*-Non-preemptive technique.

If T_i requests a lock on a data item which is already locked by T_j and if T_i has higher priority than T_j (i.e., T_i is younger than T_j), then T_i is permitted to wait. If T_i is younger than T_j, then T_i is aborted ("dies") and restarts with the same timestamp.

"It is better always to restart the younger transaction."

- *Wound-Wait*-Preemptive counterpart to wait-die.

Timestamp deadlock prevention schemes (cont'd)

Assume T_i requests lock on a data item which is already locked by T_j. If it is younger than T_j, then T_i is permitted to wait. If it is older T_j, then T_j is aborted and the lock is granted to T_i.

"Allow older transactions to pre-empt younger ones and therefore only younger transactions wait for older ones."

IMPLEMENTATION APPROACHES

- Centralized

  Periodically (e.g., few minutes) each schedule sends its local wait-for graph to the deadlock detector. The deadlock detector combines the local graphs into a system wide wait-for graph by constructing the union of the local graphs.

- Hierarchical

  The data base sites are organized into a hierarchy (or tree), with a deadlock detector at each node of the hierarchy. Deadlocks local to a single site are detected at that site. Deadlock involving two or more sites are detected by the regional deadlock detector, and so on.

Deadlock Detection in Distributed Systems.

Local wait - for graphs are not sufficient to characterize all deadlocks in the distributed systems. Instead, local wait-for graphs must be combined into a more global wait-for graph. Centralized 2PL does not have this problem since there is only one lock schedule. However, in the case of a distributed lock schedule, the coordination task becomes very complex.

Periodic transmission of a local wait-for graph can cause the following two problems:

1.) Deadlock may not be detected right away

2.) Phantom deadlock - Transaction T may restart other than concurrency control (e.g. its site crashed). Until T's restart propagates to the deadlock detector, the deadlock detector can find a cycle in the wait-for graph that includes T.

Deadlock Avoidance Method Used in Practice

- In the absence of a general analysis which determines the tradeoff of various deadlock avoidance methods, as well as the lack of understanding of complex interrelationships of deadlock among protocol levels, the commercial systems and real life applications are mainly based in the most experienced method: 2-phase locking with time-out based deadlock detection.

RESILIENT COMMIT PROTOCOL

IF AN UPDATE IS POSTED AT ANY OPERATING SITE ALL OTHER OPERATING SITES THAT KEEP A COPY OF THE FILE WILL EVENTUALLY RECEIVE THE UPDATE REGARDLESS OF MULTIPLE FAILURES.

FOR LOOSELY COUPLED SYSTEMS

CONVENTIONAL TECHNIQUE

- TWO PHASE COMMIT PROTOCOL HAS BLOCKING PROBLEM WHEN COORDINATOR FAILS.
- THREE PHASE COMMIT PROTOCOL NONE BLOCKING, BUT TOO TIME CONSUMING

THEREFORE, THESE TECHNIQUES ARE NOT SUITABLE FOR REAL TIME SYSTEM APPLICATIONS.

A LOW-COST COMMIT PROTOCOL

ASSUMPTIONS

- NO NETWORK PARTITION
- NO RELIABLE NETWORK (E.G., NO LOSS OF MESSAGES, NO OUT OF SEQUENCE MESSAGES)
- “I AM ALIVE” MESSAGES ARE PERIODICALLY EXCHANGED AMONG THE SITES FOR FAILURE DETECTION (IN LIEU OF TIME OUT AND ACKNOWLEDGMENT MESSAGES)
- FAILURE SITES ARE NOT ALLOWED TO REJOIN THE SYSTEM DURING A MISSION TIME (COULD BE RELAXED)
- ALL THE UPDATES REQUIRED FOR A TRANSACTION RESIDE AT THE COORDINATOR SITE

COMMIT PROTOCOL PROCEDURES

- FOR EACH FILE, SITES ARE NUMBERED AND UPDATES ARE SENT IN THIS SEQUENCE NUMBER.
- UPDATES ARE POSTED IMMEDIATELY AFTER BEING RECEIVED
- EACH SITE SAVES THE LAST UPDATES FROM ALL OTHER SITES
- WHEN A SITE FAILURE IS DETECTED, THE SMALLEST NUMBERED SURVIVING SITE RETRANSMITS THE LAST UPDATE RECEIVED FROM THE FAILURE SITE IN THE NUMBERED SEQUENCE.
- UPDATE SEQUENCE NUMBER IS USED TO DETECT DUPLICATES.

REDUCING MESSAGES FOR FAILURE RECOVERY

- A COORDINATOR SEND AN UPDATE COMPLETE (UC) MESSAGE TO THE SMALLEST NUMBERED SITE AFTER COMPLETING AN UPDATES BROADCAST.
- WHEN A SITE RECEIVES THE UC MESSAGE, IT DISCARDS THE SAVED UPDATE.
- THIS WILL ELIMINATE UNNECESSARY RETRANSMISSION OF COMPLETED UPDATES.

RESILIENT EWP OPERATION

- SITES ARE NUMBERED AND THE SITE WITH THE SMALLEST NUMBER IS SELECTED AS THE EW.
- EW SENDS AN UPDATE TO OTHER SITES *IN THE NUMBER SEQUENCE (I.E., LOWEST NUMBERED SITE FIRST, HIGHEST NUMBERED SITE LAST).*
- EACH NON-EW SITE SHOULD SAVE THE LAST UPDATE RECEIVED FROM THE EW.
- WHEN EW FAILS, THE SITE WITH THE NEXT SMALLEST NUMBER BECOMES THE NEW EW AND RETRANSMITS THE LAST UPDATE RECEIVED FROM THE OLD EW *IN THE NUMBER SEQUENCE.*

RESILIENT PSL OPERATION

- SITES ARE NUMBERED, ASSIGN THE SITE WITH THE SMALLEST NUMBER AS THE *PS*.
- UPDATES ARE BROADCAST IN THE NUMBER SEQUENCE.
- EACH SITE SAVES THE LAST UPDATES FROM ALL OTHER SITES.
- WHEN A *NON-PS* FAILURE IS DETECTED
  - IF THE FAILED SITE IS HOLDING A LOCK, THE LOCK IS RELEASED BY THE *PS*.
  - IF THE FAILED SITE HAS MADE A LOCK-REQUEST THE LOCK-REQUEST IS DISCARDED BY THE *PS*.
  - OTHERWISE, THE *PS* BROADCASTS THE LAST UPDATE RECEIVED FROM THE FAILED SITE IN THE NUMBER SEQUENCE.

- WHEN THE *PS* FAILS:
  - THE SITE WITH THE NEXT SMALLEST NUMBER BECOMES THE NEW *PS.*
  - TJE NEW *PS* BROADCASTS THE LAST UPDATE RECEIVED FROM THE OLD *PS* IN THE NUMBER SEQUENCE.
  - TO RESUME LOCK MANAGEMENT, THE NEW *PS* REQUESTS THE LOCK-STATUS OF OTHER SITES

SITE RECOVERY

- THE RECOVERING SITE UNDOES THE LAST UPDATE ALONG WITH THE SN AND BROADCASTS “I AM UP” MESSAGE.
- THE RECOVERING SITE IS GIVEN THE SITE NUMBER LARGER THAN ANY SURVIVING SITE NUMBER.
- AN OPERATING SITE IS SELECTED TO PROVIDE ALL LOST UPDATES FOR THE RECOVERING SITE.
- THE POSTING OF NEWLY INCOMING UPDATES (AT THE RECOVERING SITE) IS POSTPONED UNTIL ALL LOST UPDATES ARE RECEIVED.

SUMMARY

- UPDATE BROADCAST IN ONE PHASE ACCORDING TO A PREASSIGNED SITE SEQUENCE.

  PARAMETER: FREQUENCY OF “I AM ALIVE” MESSAGE.
- REQUIRES ADDITIONAL OVEHEAD ONLY WHEN FAILURE OCCURS
- THE RESILIENT COMMIT PROTOCOL CAN BE INCORPORATED INTO CONCURRENCY CONTROL TECHNIQUES (E.G., PSL,EWP).
- THE RESILIENT COMMIT PROTOCOL IS SUITABLE FOR REAL TIME APPLICATIONS.

A QUERY PROCESSING EXAMPLE FOR A DISTRIBUTED DATABASE SYSTEM

Database (suppliers, parts, and supply):

S	(S#, CITY)	10,000tuples, stored at site A
P	(P #, COLOR)	100,000 tuples, stored at site B
SP	(S#, P#)	1,000,000 tuples, stored at site A

Assume that every tuple is 100 bits long.

Query: *suppliers numbers for London suppliers of red parts*

```
SELECT   S.S. #
FROM     S, SP, P
WHERE    S.CITY = 'LONDON'
  AND    S.S.# = SP.S#
  AND    SP.P# = P.P#
  AND    P.COLOR = 'RED'
```

A QUERY PROCESSING EXAMPLE FOR A DISTRIBUTED DATABASE SYSTEM (CONT'D)

Estimates (cardinalities of certain intermediate results):

Number of red parts = 10
Number of shipments by London suppliers = 100,000

Communication assumptions:

Data rate = 10,000 bits per second
Access delay = 1 second

T [i] = total access delay + (total data volume/ date rate)
= (number of message * 1) + (total number of bits / 10,000)
(measured in seconds).

COMMUNICATION TIME FOR SELECTED DISTRIBUTED QUERY PROCESSING STRATEGIES

Strategy	Technique	Communication Time
1	Move P to A	16.7 min
2	Move S and SP to B	28 hr
3	For each London shipment check corresponding part	2.3 day
4	For each red part, check for London supplier	20 sec
5	Move London shipments to B	16.7 min
6	Move red parts to A	1 sec

DISTRIBUTED QUERY PROCESSING PROBLEM

- GIVEN A QUERY THAT REFERENCES INFORMATION STORED IN SEVERAL DIFFERENT SITES:

    1. DECOMPOSE IT INTO A SET OF SUBQUERIES OR OPERATIONS TO BE PERFORMED AT INDIVIDUAL SITES.

    2. DETERMINE THE SITE FOR PERFORMING EACH OPERATION

COST OF A QUERY PROCESSING POLICY DEPENDS ON:

– VOLUME OF DATA TRAFFIC,
– SEQUENCE OF OPERATIONS,
– DEGREE OF PARALLELISM,
– SITES OF OPERATIONS.

QUERY TREE

- A QUERY TREE REPRESENTS EACH SEQUENCE OF OPERATION THAT PRODUCES THE CORRECT RESULT.

- GIVEN AN ARBITRARY QUERY TREE, A SET OF EQUIVALENT QUERY TREES CAN BE GENERATED USING THE COMMUTATIVITY, ASSOCIATIVITY, AND DISTRIBUTIVITY PROPERTIES OF QUERY OPERATIONS.

PROPERTY OF QUERY OPERATIONS

- COMMUTATIVITY
- ASSOCIATIVITY
- DISTRIBUTATIVITY

UNARY OPERATIONS (e.g., SELECTION, PROJECTION)

BINARY OPERATIONS (e.g., JOIN, UNION, INTERSECTION, DIFFERENCE, DIVISION)

ADJACENT UNARY OPERATIONS

ADJACENT BINARY OPERATIONS

ADJACENT UNARY AND BINARY OPERATIONS

PLACEMENT OF UNARY OPERATION IN A QUERY TREE

THEOREM 1

PLACING EACH UNARY OPERATION AT THE LOWEST POSSIBLE POSITION IN A QUERY TREE IS A NECESSARY CONDITION TO OBTAIN THE OPTIMAL QUERY PROCESSING POLICY.

COROLLARY

IF THE OPTIMAL PLACEMENT OF A UNARY OPERATION IS ADJACENT TO TWO BINARY OPERATIONS, THEN A NECESSARY CONDITION TO OBTAIN THE OPTIMAL QUERY PROCESSING POLICY IS TO PROCESS THE UNARY OPERATION AT THE SAME SITE AS THE BINARY OPERATION THAT HAS THE LOWER POSITION IN THE TREE (i.e., PROCESSED EARLIER).

QUERY PROCESSING GRAPH

- 1. SEQUENCE OF OPERATIONS

  2. GROUPS OF OPERATIONS PERFORMED AT A SINGLE SITE

- STORAGE NODES HAVE NO INPUTS, AND REPRESENT INITIAL OPERATIONS ON FILE

- EXECUTION NODES HAVE ONE OR MORE INPUTS AND REPRESENT MULTI-FILE OPERATIONS.

THEOREM 3

FOR A GIVEN QUERY PROCESSING GRAPH, IF THE UNIT COMMUNICATION COSTS AMONG DIFFERENT PAIRS OF COMPUTERS ARE THE SAME, AND THE PROCESSING COSTS FOR A GIVEN OPERATION ARE THE SAME FOR ALL THE COMPUTERS, THEN FOR EACH EXECUTION NODE OF THE GRAPH, SELECTING THE STORAGE NODE SITE THAT SENDS THE LARGEST AMOUNT OF DATA TO THAT EXECUTION NODE AS THE SITE FOR PERFORMING ITS OPERATIONS YIELDS MINIMUM OPERATING COST FOR THAT GRAPH.

THEOREM 4

IF THE UNIT COMMUNICATION COSTS AMONG DIFFERENT PAIRS OF COMPUTERS IN THE DISTRIBUTED DATABASE ARE THE SAME, AND THE PROCESSING COST FOR A GIVEN OPERATION IS THE SAME FOR ALL THE COMPUTERS, IF THE SITE SELECTION FOR A GIVEN QUERY PROCESSING GRAPH BASED ON THEOREM 3 IS INCONSISTENT, THEN THAT GRAPH CAN BE REDUCED TO A SIMPLER GRAPH (ONE WITH A LESSER NUMBER OF NODES) WHICH HAS THE SAME SEQUENCES OF OPERATIONS. FURTHER, THE REDUCED GRAPH YIELDS LOWER OPERATING COST THAN THAT OF THE ORIGINAL GRAPH.

THEOREM 5

IF A QUERY PROCESSING GRAPH HAS AN EXECUTION NODE THAT HAS NO ADJACENT STORAGE NODES, THEN THIS GRAPH CANNOT PRODUCE THE OPTIMAL QUERY PROCESSING POLICY.

THEOREM 6

FOR A GIVEN QUERY PROCESSING GRAPH THAT CONTAINS A MULTI-OPERATION EXECUTION NODE CONSISTING OF A SET OF OPERATIONS (

), THE SEQUENCE OF OPERATIONS FROM THIS SET WHICH HAS LEAST PROCESSING COST IS USED BY THE POLICY THAT HAS LEAST OPERATING COST FOR THIS GRAPH.

COROLLARY

THEOREM 6 IS TRUE FOR A QUERY PROCESSING GRAPH WITH MORE THAN ONE MULTI-OPERATION EXECUTION NODE.

THEOREM 7

IF THE PROCESSING COST FOR A GIVEN OPERATION IS THE SAME FOR ALL THE COMPUTERS IN THE DISTRIBUTED DATABASE AND, FURTHER, IF THE SEQUENCE OF OPERATIONS FOR PROCESSING THE OPERATION IS FIXED, THEN THE PROCESSING POLICY THAT MINIMIZES THE COMMUNICATION COST (TOTAL VOLUME OF TRAFFIC FOR THE CASE OF THE COMMUNICATION COST AMONG EACH PAIR OF COMPUTERS BEING EQUAL) YIELDS THE LOWEST OPERATING COST AMONG THE SET OF POLICIES THAT USES THIS FIXED SEQUENCE OF OPERATIONS.

- QUERY TREE OPTIMIZATION OF UNARY OPERATIONS THEOREM 1
- COMPLETE GENERATION OF LOCAL OPERATION GROUPS FROM THE QUERY TREE: THEOREM 2
- SITE SELECTION: THEOREM 3,4,&5 TOGETHER WITH FILE ALLOCATION INFORMATION
- COMPUTATION REDUCTION: THEOREM 6
- LOCAL OPTIMAL QUERY POLICIES: THEOREM 7

PROCEDURE FOR FINDING THE OPTIMAL QUERY PROCESSING POLICY

- DECOMPOSE QUERY INTO OPERATIONS.
- GENERATE THE SET OF EQUIVALENT QUERY TREES USING PROPERTIES OF QUERY OPERATIONS AND THEOREM 1
- GENERATE QUERY PROCESSING GRAPHS FROM THE QUERY TREES (USING THE ICM).
- SITE SELECTION USING THEOREMS 3,4 & 5.
- ELIMINATE CERTAIN GRAPHS USING THEOREM 6.
- COMPUTE COMMUNICATION COST FOR EACH GRAPH.
- SELECT LOCAL OPTIMAL POLICIES BASED ON THEOREM 7
- SELECT GLOBAL OPTIMAL POLICY.

EXAMPLE

- GENERATE A LISTING OF < PART NUMBER, SUPPLIER NAME, QUANTITY > FOR ALL “WHEELS” PRODUCED IN LOS ANGELES IN A QUANTITY GREATER THAN 1,000 BY ANY ONE SUPPLIER.

QUERY PROCESSING WITH DOMAIN SEMANTICS

Wesley W. Chu

Computer Science Department
University of California
Los Angeles

QUERY OPTIMIZATION PROBLEM

To find a sequence of operations which has the minimal processing cost.

CONVENTIONAL QUERY OPTIMIZATION (CQO)

For a given query:

- Generate a set of query that are equivalent to the given query.
- Determine the processing cost of each such query.
- Select the lowest cost query processing strategy among these equivalent query.

LIMITATIONS OF CQO

- There are certain queries that cannot be optimized by Conventional Query Optimization.

- For example, given the query:

  "Which ships have deadweight greater than 200 thousand tons?"

search of entire database may be required to answer this query.

THE USE OF KNOWLEDGE

- ASSUMING EXPERT KNOWS THAT:

1. SHIP relation is indexed on *ShipType*. There are about 10 different ship types, and
2. the ship must be a "SuperTanker" (one of the ShipTypes) if the deadweight is greater than 150K tons.

- AUGMENTED QUERY:

  "Which SuperTanker have deadweight greater than 200K tons?"

- RESULT:

  About 90% time saved in searching the answers.

The technique of improving queries with semantic knowledge is called *Semantic Query Optimization.*

SEMANTIC QUERY OPTIMIZATION (SQO)

Uses domain knowledge to transform the original query into a more efficient query yet still yields the same answer.

Assuming a set of integrity constraints is available as the domain knowledge,

- Represent each set of integrity constraints as P⇨ C_i, where $1 \leq i \leq n$.
- Translate (Augment) original query Q into Q' subject to C_1, C_2, ...,C_n, such that Q' yields lower processing cost than Q.
- Query Optimization Problem: Find $C_1, C_2, \ldots, C_m$ that yields minimal query processing cost; that is, C(Q') = min C(Q C_1 ... C_m)

SEMANTIC EQUIVALENCE

Domain knowledge of the database application maybe used to transform the original query into *semantically equivalent* queries.

Semantic Equivalence:

Two queries are considered to be semantically equivalent if they result in the same answer in any state of the database that conforms to the *Integrity Constraints.*

Integrity Constraints:

A set if rules that enforce the database to be accurate instance of the real world database application.

SEMANTICAL EQUIVALENCE(cont'd)

Integrity Constraints:

- *state snapshot constraints*

  e.g., if deadweight ≥ 150K then ShipType = "SuperTanker."

- *state transition constraints:*

  e.g., salary can only be increased, i.e., salary (new) ≥ salary (old).

LIMITATIONS OF CURRENT APPROACH

Current approach of SQO using :

- Integrity constraints as knowledge
- Conventional data models

LIMITATION OF INTEGRITY CONSTRAINTS

- Integrity constraints are often too general to be useful in SQO, because:
  - integrity constraints describe every possible database state
  - user is only concerned with the current database content.
- Most database do not provide integrity checking due to :
  - unavailability of integrity constraints
  - overhead of checking the integrity.

Thus, the usefulness of integrity constraints in SQO is quite limited.

LIMITATIONS OF CONVENTIONAL DATA MODELS

Conventional data models lack expressive capability for modeling conveniences. Many useful semantics are ignored. Therefore, limited knowledge are collected.

FOR EXAMPLE:

"Which employee earn more than 70K a year?"

the integrity constraint:

"The salary range of employee is between 20K to 90K."

is useless in improving this query.

AUGMENTATION OF SQO WITH SEMANTIC DATA MODELS

If the employees are divided into three categories: MANAGERS, ENGINEERS, STAFFS, and each category is associated with some constraints:

1. The salary range of MANAGERS is from 35K to 90K.
2. The salary range of ENGINEERS is from 25K to 60K.
3. The salary range of STAFF is from 20K to 35K.

A better query can be obtained:

"Which managers earn more than 70K a year?"

SUMMARY

Contributions:

Providing a model-based methodology for acquiring knowledge from the database by rule induction.

Applications:

1. Semantic Query Processing -

   use semantic knowledge to improve query processing performance

2. Deductive Database Systems -

   use induced rules to provide intensional answers.

3. Data Inference Applications -

   use rules to improve data availability by inferring inaccessible data from accessible data.

DATABASE SEMANTICS

Database semantics can be classified into:

- *database structure*, which is the description of the interrelationships between database objects.
- *database characteristics*, which defines the characteristics and properties of each object type.

However, only tools for modeling database structure are available. Very few tools exist in gathering and maintaining the database characteristics.

KNOWLEDGE ACQUISITION

A major problem in the development of a knowledge-based data processing system.

- *Knowledge Engineers* - persons in the use of expert system tools.
- *Domain Experts - persons with the expertise of the application domain.*

The Process:

- Studying literature to obtain fundamental background.
- Interacting with domain experts to get their expertise.
- Translating the expertise into knowledge representation.
- Refining knowledge base through testing and further interacting with domain experts.

A VERY TIME-CONSUMING TASK!

KNOWLEDGE ACQUISITION DATABASE

- database schema is defined according to database semantics, and
- database instances are constrained by the database charateristics,

Thus,

- database characteristics can be induced as the semantic knowledge from the database
- database schema can be a useful tool to guide the knowledge acquisition.

KNOWLEDGE ACQUISITION BY RULE INDUCTION

Given and object hierarchy and a set of database instances contained in the object hierarchy, a set of classification rules can be induced by inductive learning techniques .

Given:

H - an object type hierarchy : H_1,....,H_n

S - object schema

I - database instances representing *H*

Find:

D - a set of descriptions, D_1,..., D_n such that

for all *x*, *x* in *I* ,

if $D_i(x)$ is true, then *x* ISA H_i .

KNOWLEDGE ACQUISITION BY RULE INDUCTION (CONT'D)

Example:

SUBMARINES **contains** SSN, SSBN

D_{SSN} : $2145 \leq \text{Displacement} \leq 6955$

D_{SSBN} : $7250 \leq \text{Displacement} \leq 30000$

MODEL-BASED KNOWLEDGE ACQUISITION METHODOLOGY

The methodology consists of:

- a Knowledge-based ER (KER) Model,
- a knowledge acquisition methodology, and
- a rule induction algorithm.

KER is used as a knowledge acquisition tool when

  - no knowledge specification is provided, or
  - the database already exists.

KNOWLEDGE-BASED (KER) MODEL

To capture the database characteristics, a Knowledge-based Entity Relationship (KER) is proposed to extend the basic ER model to provide knowledge specification capability.

A KER schema is defined by the following constructs:

1. **has-attributed/ with** *(aggregation)*
   This construct links an object with other objects and specify certain properties of the object.
2. **isa/with** *(generalization)*
   This construct specifies a type/subtype relationship between object types.

KNOWLEDGE-BASED ER (KER) MODEL (CONT'D)

3. **has-instance** *(classification)*

This construct links a type to an object that is an instance of that type.

The knowledge specification is represented by the *with-constraint* specification.

CLASSIFICATION OF SEMANTIC KNOWLEDGE

Domain knowledge:

Specifying the static properties of entities and relationships.

e.g., displacement in the range of (0 - 30,000)

Intra-Structure knowledge:

Specifying the relationships between attributes within an object (an entity or a relationship).

e.g., if the displacement is less than 7000, then it is a nuclear submarine.

Inter-Structure knowledge:

Specifying the relationship that is related to attributes of several entities of the aggregation relationship.

CLASSIFICATION OF SEMANTIC KNOWLEDGE (CONT'D)

Inter-Structure knowledge:

Specifying the relationship that is related to attributes of several entities of the aggregation relationship.

e.g., the instructor's department must be the same as the department of the class offered.

KNOWLEDGE ACQUISITION METHODOLOGY

To provide a systematical way of collecting domain knowledge guided by the database schema. It consists of three steps:

- Schema Generating - using KER.
  a. Identify entities and associated attributes.
  b. Identify type hierarchies by determining the class attributes of each type hierarchy.
  c. Identify aggregation relationships. Define each referential key as a class attribute.

- Rule Induction.
- Knowledge Base Refinement.

RULE INDUCTION ALGORITHM

Semantic rules for pairwise attributes (X --> Y) are induced using the relational opaerations.

Sketch of the Algorithm:

1. *Retrieving (X,Y)* value pairs.

   Retrieve the instance of the *(X,Y)* pair from the database. Let *S* be the result.

2. *Removing inconsistent (X,Y)* value pairs.

   Retrieve all the *(X,Y)* pairs that for the same value of *X* has multiple values of *Y*. Let *T* be the result.

   Let $S = S - T$.

RULE INDUCTION ALGORITHM (CONT'D)

3. *Constructing Rules.*

For each distinct value of Y in S, say y, determine the value range x of X and create a rule in the form of

if $x_1 \leq X \leq x_2$ then Y=y.

EXAMPLES OF INDUCED RULES

A prototype system was implemented at UCLA using a naval ship database as a test bed. Examples of rules induced are :

Entity: SUBMARINE

x **isa** SUBMARINE

R_1 : **if** 0101 $\leq$ x.*Class* $\leq$ 0103 **then** x **isa** SSBN

R_2 : **if** 0201 $\leq$ x.*Class* $\leq$ 0215 **then** x **isa** SSN

R_3 : **if** Skate $\leq$ x.*ClassName* $\leq$ Thresher **then** x **isa** SSN

R_4 : **if** 2145 $\leq$ x.*Displacement* $\leq$ 6955 **then** x **isa** SSN

R_5 : **if** 7250 $\leq$ x.*Displacement* $\leq$ 30000 **then** x **isa** SSBN

EXAMPLES OF INDUCED RULES (CONT'D)

Relationship: INSTALL

x **isa** SUBMARINE and y **isa** SONAR

R_1: **if** SSN582 ≤ x.*Id* = SSN601 **then** y **isa** BQS
R_2: **if** SSN604 ≤ x.*Id* = SSN671 **then** y **isa** BQQ
R_3: **if** **x.***Class* = 0203 **then** y **isa** BQQ
R_4: **if** 0205 ≤ x.*Class* ≤ 0207 **then** y **isa** BQQ
R_5: **if** 0208 ≤ x.*Class* ≤ 0215 **then** y **isa** BQS
R_6: **if** y.*Sonar* = BQS-04 **then** x **isa** SSN

PRUNING THE RULE SET

When the number of rules generated becomes too large, the system must reduce the size of the knowledge base.

Two Criteria for Rule Pruning:

1. Coverage.

   Keep the rules that are satisfied by more than N_c instances and drop those rules that are satisfied by less than N_c instances.

2. Completeness.

   Keep the rule schema (X --> Y) that the total number of instances satisfied by the rules of the same scheme is greater than a coverage threshold C_c.

SUMMARY

- Contirbutions:

  Providing a model-based methodology for acquiring knowledge from the database by rule induction.

- Applications:

  1. Semantic Query Processing -
     use sematic knowledge to improve query processing performance.
  2. Deductive Database Systems -
     use induced rules to provide intensional answers.
  3. Data Inference Applications -
     use rules to improve data availability by inferring inaccessible data from accessible data.

Generate the Rules

- Select *targets*

  Targets are the RHS attributes of rules.

  Method of selection.

    - Use indices as targets
    - Use selectivity

      selectivity = # of tuples with disticnt value / total # of tuples
    - Targets are chosen based on database schema (e.g. type hierarchy).
- Generate rules for each target

FAULT TOLERANT DDBMS VIA DATA INFERENCE

Network Partition

Causes: failures of

- **channels**
- **nodes**

Effects: - queries cannot be processed if the required data is inaccessible.

- replicated files in different partitions may be inconsistent.

- updates may only be allowed in one partition.

- transactions may be aborted.

Conventional Approach for Handling Network Partitioning

- **Based on *syntax* to serialize the operations**
- **To ensure data consistency**
  - **Not all queries can be processed**
  - **Based on data availability, determine which partition is allowed to perform database update.**

Poor Availability!

New Approach

- **Exploit data and transaction *semantic***

  – **Use Data Inference Approach**
    - **Assumption: Data are correlated**

      e.g.

      salary and rank

      ship type and weapon
    - **Infer inaccessible data**

  – **Use semantic information to permit update under network partitioning**

Query Processing System with Data Inference

- Consists of
  - DDBMS
  - Knowledge base (rule based)
  - Inference engine

Motivation of Open Data Inference

- Correlated knowledge is incomplete
  - Incomplete rules
  - Incomplete objects

Example of Incomplete Object

Type -----> Weapon

IF type in {CG, CGN} THEN weapon = SAM01
IF type = DDG THEN weapon = SAM02

TYPE	WEAPON
CG	SAM01
CGN	SAM01
DDG	SAM02
SSGN	??

Result: Incomplete rules generate incomplete object

Merge of Incomplete Objects

Observation:

- Relational join is not adequate for combining incomplete objects
    - lose information

Questions:

- What kind of algebraic tools do we need to combine incomplete objects without losing information?
- Any correctness criteria to evaluate the incomplete results?

Merge of Incomplete Objects

TYPE ---> WEAPON and WEAPON --->WARFARE

Type	Weapon	Weapon	Warfare
CG	SAM01	SAM01	WF1C
CGN	SAM01	SAM03	WF1D
DDG	SAM02		
SSGN	?		

Use relational join to combine the above two paths :

Type	Weapon	Warfare
CG	SAM01	WF1C
CGN	SAM01	WFIC

Merge of Incomplete Objects (Cont'd)

Other way to combine:

TYPE	WEAPON	WARFARE
CG	SAM01	WF1C
CGN	SAM 01	WF1C
DDG	SAM02	?
?	SAM03	WF1D
SSGN	?	?

New Algebraic Tools for Incomplete Objects

- S-REDUCTION
  - Reduce redundant tuples in the object
- OPEN S-UNION
  - Combine incomplete objects

S-Reduction

- Remove redundant tuples in the object
- Object RR with key attribute A is reduced to R

RR

A	B	C
a	1	aa
b	2	_
c	_	cc
a	1	aa
b	_	bb
c	_	_

R

A	B	C
a	1	aa
b	2	bb
c	_	cc

Open S-Union

- Modify join operation to accommodate oncomplete information

- Used to combine closed/open objects

R1		R2		---->	R		
sid	type	type	weapon		sid	type	weapon
s101	DD	DD	SAM01		s101	DD	SAM01
s102	DD	CG	-		s102	DD	SAM01
s103	CG				s103	CG	-

Open S-Union and Toleration

- Performing open union on two objects R1, R2 generates the third object which tolerates both R1 and R2

R1		R2		---->	R		
sid	type	type	weapon		sid	type	weapon
s101	DD	DD	SAM01		s101	DD	SAM01
s102	DD	CG	-		s102	DD	SAM01
s103	CG				s103	CG	-

- R tolerates R1
- R tolerates R2

Implementation

Derive missing relations from accessible relations and correlated knowledge

Three types of derivations:

- View mechanism to derive new relations based ib cretain source relations
- Valuations of incomplete relations based on correlated knowledge
- combine two intermediate results via open s-union operation

Example of Open Inference

DERIVATION 1: select sid, type from SHIP, CLASS

DERIVATION 2: CLASS (type) --> INSTALL (weapon)

R1		R2		------>	INSTALL_INF		
sid	type	type	weapon		sid	type	weapon
sid	DD	DD	SAM01		s101	DD	SAM01
s102	DD	CG	-		s102	DD	SAM01
s103	CG				s103	CG	-

- **INSTALL_INF can be used to replace missing relation INSTALL**

FAULT TOLERANT DDBMS VIA INFERENCE TECHNIQUES

- Query Processing Under Network Partitioning
    - Open Inference: Inference with incomplete information
    - Algebraic tools for manipulating incomplete objects
    - Toleration: weaker correctness criteria for evaluating incomplete information

CONCLUSION

Data Inference is an effective method for providing database fault tolerance during network partitioning.

CoBase

Objective

Status

- CoBase Architecture
- Relaxation Mechanism
- Implementation
- Data Inference with Complete Measure

Continuing Work

New Direction

Demo

Cooperative Distributed Database Systems

The Next generation of distributed database systems, using Inference techniques to provide:

- Fault Tolerance Capabilities
- Cooperative Query Answering

Validate concepts with prototype cooperative distributed DDBMS

Cooperative Query Answering

Motivation

Conventional query answering

- provides too much data
- may obscure real meaning of data
- time consuming
- needs to know the detailed database schema
- cannot get approximate answer if full data is unavailable
- cannot analyze the intent of the user and derive relevant information if the exact answer is not available
- cannot answer conceptual queries

Cooperative Query Answering

Derive Summary Answers

Derive Intensional Answers

Derive Approximate Answers

Answer Conceptual Queries

New Methodologies for Cooperative Query Answering

Type Inference

Neighboring Inference

Type Inference

Approach

- Use knowledge induction to analyze database contents and domain knowledge to derive a set of If- Then rules.
- Based on application domain, extend data schema with type hierarchy.
- Use induced rules to derive intensional answers from extended schema.

Application:

- Summary answer
- intensional answer

Neighborhood Inference

Approach

- Translate a query on missing data into a related query on available data
- Provide answers with different degrees of generality, coverage and approximation

Mechanisms

Based on semantic knowledge, organize data in type abstraction hierarchy

Provide multi-level knowledge representation

Support inference between different knowledge levels

- generalization
- specialization
- association

Type Abstraction Hierarchy

Abstract view of type heirarchy

Integrates:

- Subsumption (is_a)
- Composition (part_of)
- abstraction

Provides multi-level knowledge representation

Relaxation Mechanism

Relaxation Variables

- Types
- Attributes
- Attribute Values

Nearness Measure

Context Dependent

- Time
- Space
- Concept

Relaxation Specifics

Explicit (by user)

- relaxable range specified by C-SQL
- primitives (relaxable predicates)

Implicit (by system)

- generalization & specialization
- based on
  - context
  - type abstraction hierarchy
  - abstract domain values

Interactive

- through user/ system interaction

C-SQL

An extended Query Language

Primitives (Predicates) for Cooperative Query Answering

Context Free

- approximate

  ^9 AM

- inclusion

  between (7 AM, 11 AM)

- set membership

  within {‘LAX’, “Burbank”}

Context Sensitive (nearness)

- Restaurant **near-to** ‘Redondo Beach’
- Airport **near-to** ‘LAX’
- Chinese Restaurant **nearest-to** ‘UCLA’

Relaxation Order may be specified

- **relaxation-order** (food style, location)

Original Query

select * *from* delta-flight

where dep-airport = “Newark”

and arr-airport **near-to** “LAX”

Original Query

select * *from* restaurants

where type = “Chinese”

and location **near-to** “LAX”

Original Query

select * *from* american-flight

where dep-airport = “National”

and arr-airport **near-to** “LAX”

and dep-time *between* (1000 ^**1100**)

Similar-to

Find all airports in Tunisia *similar to* the Bizerte airport in terms of runaway length and (more importantly) runway width.

select *aport_name*,
runway_length_ft,, runway_width_ft
from *runways, countries*
where *aport_name* **similar-to** 'Bizerte'
based-on
((*runway_length_ft 1.0)*
(*runway_width_ft 2.0)*) and
country_state_name_long = 'Tunisia' and
and countries.glc_cd = runways.glc_cd;

Similar-to Evaluation

CoBase first evaluates the query without the similar-to clause:

select *aport_name*,
runway_length_ft, *runway_width_ft*
from *runways*, *countries*
where *country_state_name_long* = ‘Tunisia’ and
and countries.glc_cd = runways.glc_cd;

Patterns

Specified by query conditions one or more attributes

- arrival time = 10AM
- departure airport = LAX

Advantages

- Finer granularity than types
- More specific knowledge representation
- More complex queries can be derived from logical operation patterns
- Unified interface between KB and DB

Pattern: **dep-airport = “Lax”**

Pattern: **los-angeles-dep-airport**

Associated Subject: **Weather**

Information derived from association:

Adverse weather conditions over LA in winter might delay flights.

Pattern: **arr-airport = “O-Hare”**

Pattern name: **Chicago-arr-airport**

Associated Subject: **Weather**

Information derived from association: *Snowstorm over Chicago in winter might divert flights to other airports.*

Association Control

Search for Association Links

Termination

Rule Based Approach

Static Rules

Do not Consider

- User Model
- Application Context

No Termination

Case-Based Approach

- Use Past Cases to Infer Future Association
- Consider
    - User Model
    - Query and Application Context
- Termination
    - Association Ranking

The Demo with CoBase,SIMS and LIM

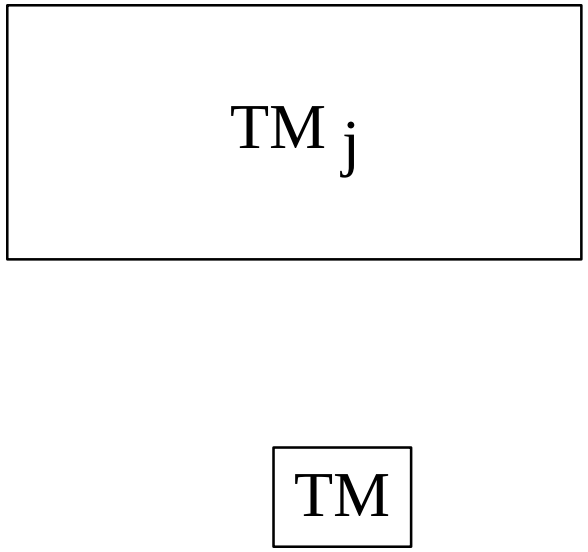
Example Query:

Find all airports **near-to** Bizerte that can land the aircraft KC-10.

CoBase - provides relaxation primitive: **near-to**

SIMS - provides access to multiple databases

LIM - provides access to a single database


TM j
TM