The Hong Kong University of Science and Technology

COMP334: Distributed Database Systems

Final Examination

16 December 1999

Note: This is an open book examination. You can consult books and notes to answer the questions, but do not simply copy from them.

Please check whether you have all the eleven pages. You are required to answer all questions in the space provided.

Your Student Number: _____

Your Name: _____

Lab: _____

Question	Total marks	You got
Q1	20	
Q2	25	
Q3	20	
Q4	15	
Q5	20	
Total	**100**	

Question 1 (20%)

(1) (4%) Dose $R \ltimes S$ necessary equal $S \ltimes R$?
Under what conditions does $R \ltimes S = S \ltimes R$ hold?

No. **(1)**

$R \ltimes S = S \ltimes R$ holds if R and S have the same schema **(3)**

Answer R=S get 1 mark

(2) (5%) Blooming join of two relations, *R* and *S* at site S_1 and S_2, works as follows. At site S_1, a bit-vector of some size *k* is computed by hashing the join attribute values of *R* into range 0 to *k*-1 and set bit *I* ($0 \leq I < k$) to 1 if some tuple hashes to *I*. After the bit-vector is computed, it is shipped to S_2. At site S_2, each tuple of *S* is hashed similarly. If the corresponding bit in the *R*'s bit-vector is 0, the tuple is discarded. The remaining tuples of this reduction process are shipped to site S_1 to join with *R*. **Write the Blooming join algorithm in the form of psudo-code.**

```
Clear all bits in bit-vector B
foreach tuple r in R do
    begin
        I := hash (r.A);
        B[I] := 1;
    end;
Ship B to S2;

for each tuple s in S do
    begin
        I :=  hash(s.B);
        If B[I] = 1 then insert s to S'
    end
Ship S' to S1

Join R and S' at site S1
```

(3) (3%) Compared to semi-join, what are the pros and cons of blooming join?

+ bit-vector size can be smaller than the size of projection of the join attributes
- may ship some tuples that will not produce join results

(4) (8%) Give the cost formula of blooming join of relation *R* with *S*. Describe clearly the meaning of the symbols used in your formula.

One scan of relation R; Hash each tuples in R; Transfer the bit-vector
One scan of relation S; Hash each tuple of S Store/Ship relation S'
Local join cost of S' and R (S' may be stored)

Question 2 (25%)

Consider a database consisting the following two relations

Employees (***eid*: integer**, *did*: integer, *sal*: real)

Departments (***did*: integer**, *mgrid*: integer, *budget*: integer)

The *mgrid* field of Departments is the *eid* of the manager. Each of these relations contains 20-byte tuples, and the *sal* and *budget* fields both contain uniformly distributed values in the range 0 to 1,000,000. The Employees relation contains 100,000 pages, the Departments relation contains 5000 pages, and each processor has 100 buffer pages of 4000 bytes each. The cost of one page I/O is t_d and the cost of shipping one page is t_s. There are no indexes.

The database is stored in a distributed DBMS with 10 sites. The Departments tuples are horizontally partitioned across the 10 sites by did, with the same number of tuples assigned to each site, and with no particular order to how tuples are assigned to sites. The Employees tuples are similarly partitioned, by *sal* ranges, with $sal \leq 100{,}000$ assigned to the first site, $100{,}000 < sal \leq 200{,}000$ assigned to the second site, etc. In addition, the partition $sal \leq 100{,}000$ is frequently accessed and infrequently updated, and it is therefore replicated at every site. No other Employees partition is replicated.

(1) (5%) Give the cost of compute the natural join of Employees and Departments using the strategy of shipping all fragments of the smaller relation to every site containing tuples of the larger relation.

1. *We **need to do a total exchange of the Departments partitions.** In other words, each site must ship its partition to each of the other nine sites. Each of the Departments partition is stored in 500 pages. For each site, the cost of shipping a partition to the other nine sites is 500* 9t_s. Hence, the cost of the total exchange is 45,000 t_s.*

2. ***The cost of a local natural join**:*
   using Hash Join is 3(M +N)td , where M = 10,000 Departments records, and N = 10,000 pages (Employee has 100,000 pages, 10000 for each site

3. ***Finally, the result fragments have to be shipped to the query site**. Each site will have a result fragment of 500 pages (the same size as the local Departments fragment) and the total cost of shipping these fragments from nine sites to the query site is 4500ts .*

4. *Adding all the costs together, the total cost of the plan.*

Describe the best plan for the following query and give its cost.

(2) (6%) Find the highest paid employee.

1. *Clearly, the highest paid employee will have a salary in the bracket 900,000 to 1,000, 000. (We are assured that there is at least one employee in this salary range because the sal field has values uniformly distributed in the range 0 to 1,000,000, and there are 100,000 pages of Employees tuples). So we ship the query to the tenth site, evaluate the highest paid employee at that site, and ship the result back to the query site. Finding the employee(s) with the maximum salary at the tenth site requires a scan of the Employees fragment, which costs 10,000td .*

2. *Since the salary is uniformly distributed, we may assume there are 200*10,000/ 100,000 = 20 result tuples, which fit on a single page. Thus total cost of the query is 10,000td + ts .*

(3) (6%) Find the highest paid employee with salary greater than 450,000 and less than 550,000

Employees in the salary bracket between 450000 and 550; 000 are split across the fifth and the sixth sites. But clearly, by the partitioning of the Employees relation, the employees in the sixth site earn more than those at the fifth site. So we need to query only the partition in the sixth site, but we need to qualify our query by selecting only those employees whose salary is less than 550,000, and then selecting the one with the highest salary. The cost is again the same as in part (2) above.

(4) (8%) Find the highest paid manager for those departments stored at the query site

***Plan**: We need to do a partial join of the two relations. The Departments fragment at the query site is shipped across to each of the other nine sites, and the fragment is joined with the Employees fragment at the site. (Note: semijoin is no better)*
At each site, the result is scanned to select the tuple with the maximum salary, and this tuple is shipped to the query site, where a final scan of the ten tuples contributed by each site yields the department (amongst the ones stored at the query site) with the highest paid manager.

***The cost**:*
(1) the initial shipping (4500ts) ,
(2) the cost of ten local joins
(3) the cost of the scans
(5) the cost of shipping the result tuples (9ts) ,
(6) the cost of the final scan is 10td .

Question 3 (20%)

Suppose that an Employees relation is stored in Hong Kong and the tuples with sal ≤ 100,000 are replicated at Beijing. Consider the following three options for lock management:

- all locks managed at a *single site*, say Shenzheng;
- *primary copy* with Hong Kong being the primary for Employees; and
- *fully distributed*.

For each of the lock management options, explain locks are set at which site for the following queries. Also state which site the page is read from.

(1) (5%) A query submitted at Shanghai wants to read a page containing Employees tuples with *sal* ≤ 50,000.

Shanghai does not have a copy of the relevant partition. So it needs to query either the Hong Kong or Beijing databases. In the following table, wherever Hong Kong and Beijing are listed together, both possibilities exist; the choice is the city the query is shipped to.

Protocol	*Locks Set At*	*Pages Read From*
Single Site	*Shenzheng*	*Beijing/Hong Kong*
Primary Copy	*Beijing*	*Beijing/Hong Kong*
Fully Distributed	*Beijing/Hong Kong*	*Beijing/Hong Kong*

In the last case, the locks are set at, and the pages are read from the same site - depending on where the query is shipped.

(2) (5%) A query submitted at Hong Kong wants to read a page containing Employees tuples with *sal* ≤ 50,000.

Hong Kong can query its own database. So the query and the result do not have to be shipped.

Protocol	*Locks Set At*	*Pages Read From*
Single Site	*Shenzheng*	*Hong Kong*
Primary Copy	*Hong Kong*	*Hong Kong*
Fully Distributed	*Hong Kong*	*Hong Kong*

(3) (5%) A query submitted at Beijing wants to read a page containing Employees tuples with *sal* ≤ 50,000.

3. Hong Kong can query its own database. So the query and the result do not have to be shipped.

Protocol	*Locks Set At*	*Pages Read From*
Single Site	*Shenzheng*	*Beijing*
Primary Copy	*Hong Kong*	*Beijing*
Fully Distributed	*Beijing*	*Beijing*

(4) **(5%) An update transaction, Give all employees a 10% raise, is issued in the DDBS described in Question 2. Describe the sites visited and the locks obtained**. ***Hint: ROWA policy is used for replications, and the conditions of the original partitioning of Employees must still be satisfied after update.***

1. *We need to obtain an exclusive lock on all the copies of the fragment with employees whose salary is less than 100,000. The remaining fragments of the relation are not replicated, and hence, just one exclusive lock for each fragment is required, at the site where it is stored.*

2. *Once the update is complete, we need to do some shipping to satisfy the original partitioning of Employees. Each site will ship to the "next" site employees whose salaries have been raised to the next bracket of the original partitioning. In other words, the first site will ship all records will salaries greater than 100,000 to the second site, the second site will ship all records will salaries greater than 200,000 to the third site, and so on. To do this, each site needs to get an exclusive lock on the fragment they store. Once each site has modified its relation, shipped the appropriate records, and has received its set of additional records, it obtains a lock on its fragment to append the new records.*
3. *The only tricky fragment to handle is the one that has the employees with the lowest salaries, since it is replicated at all the sites. The changes made at the first site must be executed at all the sites. Hence, each site must obtain an exclusive lock on its copy of the fragment, and run the query against it.*

Question 4. (15%)

(1) (5%) Compare the relative merits of centralized and hierarchical deadlock detection in a distributed DBMS.

A centralized deadlock detection scheme is better for a distributed DBMS with uniform access patterns across sites since dead-locks occurring between any two sites are immediately identified. However, this benefit comes at the expense of frequent communications between the central location and every other site. It is often the case that access patterns are more localized, perhaps by geographic area. Since deadlocks are more likely to occur among sites with frequent communication, the hierarchical scheme will be more efficient in that it checks for deadlocks where they are most likely to occur. In other words, the hierarchical scheme expends deadlock detection efforts in correlation to their probability of occurrence, thus resulting in greater efficiency.

Consider the following modification to a local waits-for graph: Add a new node T_{ex}, and for every transaction T_i that is waiting for a lock at another site, add the edge $T_i \rightarrow T_{ex}$. Also add an edge $T_{ex} \rightarrow T_i$ if a transaction executing at another site is waiting for T_i to release a lock at this site.

(2) (4%) If there is a cycle in the modified local waits-for graph that does not involve T_{ex}, what can you conclude? If every cycle involves T_{ex}, what can you conclude?

A cycle in the modified waits for graph not involving $\mathbf{T_{ex}}$ *clearly indicates that the deadlock is internal to the site with the graph. If every cycle involves* $\mathbf{T_{ex}}$*, then there may be a multiple-site or potentially global deadlock.*

(3) (6%) Suppose that every site is assigned a unique integer site-id. Whenever the local waits-for graph suggests that there might be a global deadlock, send the local waits-for graph to the site with the next higher site-id. At that site, combine the received graph with the local waits-for graph. If this combined graph does not indicate a deadlock, ship it on to the next site, and so on, until either a deadlock is detected or we are back at the site that originated this round of deadlock detection. **Is this scheme guaranteed to find a global deadlock if one exists?**

The scheme is guaranteed to find a global deadlock provided that the dead- lock exists prior to when the first waits-for graph is sent. If this condition is met, then the global deadlock will be uncovered before any node receives a graph containing its own nodes back.

Question 5 (20%)
(1) **(3%) Explain the need for a commit protocol in a distributed DBMS**.

Crash recovery in a distributed database is complicated by new kinds of failures such as crash of a remote site or breakdown in communication links. Also, either all participants of a given transaction must commit or none of them must commmit. This all-or-none requirement forces us to use some commit protocol, to ensure all subtransactions either commit or abort.

(2) (3%) In two phase commit protocol, the participants send an *ack* message to the coordinator after receiving global-commit/global-abort message. Why do we need the *ack* messages? Are the *ack* messages necessary?

Ack messages let the coordinator know that its message has been received by the participants and the subtransaction has been committed/ aborted, as per the coordinator's instructions. It signals the end of the protocol, as well as the transaction. **(2)**

Without ack message, 2PL should work also **(1)**

(3) (6%) Suppose that a site does get any response from another site for a long time. Can it tell whether the connecting link has failed or the other site has failed? How is such a failure handled?

1. *I do not think it is possible for a site to make out if a particular communication link has failed, or the site at the other end of the link is down.*
2. *If the site is a coordinator, it can aborts its transaction, and instruct the other subordinates to abort their subtransactions.*
3. *If the site is a subordinate, it does not have the prerogative to abort the subtransaction. It is essentially blocked and must keep sending messages to the coordinator, at regular intervals, and wait for a reply.*

(4) (8%) Suppose that the coordinator includes a list of all subordinates in the prepare message. If the coordinator fails after sending out either an abort or commit message**, can you suggest a way for active sites to terminate this transaction without waiting for the coordinator to recover? Assume that some but not all of the abort/commit messages from the coordinator are lost.**

1. *After having waited for some pre-determined time, the active sites that have not received a message from the coordinator could send a message to the list of subordinates in the prepare message.*
2. *The sites that have received the coordinator's decision, and could send the decision on global commit or abort the sites that enquired.*
3. *A subordinates could commit/abort their subtransactions accordingly.They do not have to send an ack message to the coordinator.*
4. *When the coordinator comes back up, the recovery module can determine the transaction has been committed by the coordinator and all the subordinates, and enter an end log record for the transaction at the coordinator site.*