Part A
DB Qual Exam – Spring 2003

Basic Part (20 points)

Problem A.1 (10 points)

Consider a schedule S_1 with the following arcs in its precedence graph (and only these arcs): $T_1 \rightarrow T_2$, $T_3 \rightarrow T_2$, $T_1 \rightarrow T_3$, $T_4 \rightarrow T_1$.

(a) Give a serial schedule that is equivalent to S_1.

(b) Consider a second schedule S_2 that we know is conflict-equivalent to S_1. What is the precedence graph of S_2?

(c) Let $Sh(T_1)$ be the shrink point of T_1 in S_1, i.e., the first unlock action of T_1 in S_1. If S_1 is a legal schedule, and all transactions are well formed and two-phase, what can we say about $Sh(T_2)$ and $Sh(T_4)$ with respect to $Sh(T_1)$?

(d) Consider a third schedule S_3 whose precedence graph happens to have exactly the same vertices and arcs given above. Is S_3 conflict-equivalent to S_1?

Problem A.2 (10 points)

Consider a B+ tree of order $n = 3$ (each block has n keys), with $l = 3$ levels. Assume that all blocks are full.

(a) How many records do the leaves of this particular B+ tree point to? Call this result R.

(b) How many blocks (nodes) are there in this particular tree? Call this result K.

(c) If the disk fails and we lose one B+ tree block, we lose access to some of the records. For example, if we lose the root, we lose access to *all* the records (unless there is some other access path). If we lose a leaf block, then we only lose access to n records. (Ignore the links that connect leafs in a chain for this problem.) For this problem, compute the expected number of records that will be lost due to a *single* B+ block loss. Assume that any tree block is equally likely to fail, and assume there are no access paths other than our B+ tree.

(d) What is the general expression for expected number of lost records due to a single block failure, as a function of n, l, and K and R?

(e) What is the general expression for R in terms of n and l?

(f) What is the general expression for K in terms of n and l?

Advanced Part (40 points)

Problem A.3 (15 points)

(a) When building a text index, as a first step the text is parsed into a sequence of triplets and written to a temporary file. What does each triplet contain?

(b) The seqeunces of triplets are often compressed. Typically, how large is the compressed version compared to the uncompressed version? Just give a rough factor.

(c) In your readings you studied several algorithms for constructing inverted indexes. Which *two* are best suited for very large collections of text? Explain briefly why.

(d) One of the papers you read discussed the challenge of "making application logic a first class citizen in future database systems." What does this phrase mean? In your own words, what is this challenge?

(e) What does the term "no knobs operation" refer to? Please explain briefly.

Problem A.4 (25 points)

For this problem we want to design an extensible hash table whose buckets (blocks) can be distributed across several servers. Here is the basic idea:

- There are N servers in the system.
- The hash table supports lookup(key) and insert(key, value). Assume no deletions for this problem.
- We are implementing extensible hashing. A bucket can be at any one of the N nodes. Thus, a bucket address is given by the pair (j, d) where j is the server id, and b is the location on disk.
- The directory for the extensible hash table is replicated at all servers. Each node j keeps a local variable $i(j)$ indicating how many hash bits are in use. Thus, the directory copy at node j will contain $2^{i(j)}$ entries.
- Replication is done in a lazy fashion, so the directories may not be consistent or up to date.
- If a directory entry $X = (i, b)$ at node i points to a disk block on node i, then the entry is up-to-date. That is, entry X is the "master copy" in this case.

Here is a simple example to illustrate how this scheme should work. Say a user issues a query for key k at node j. Node j hashes k and does a lookup in the local directory, in the normal (for extensible hashing) fashion. Say the entry found is $X = (j, b)$. Then we know the key we want is in local block b and we can complete the lookup. Instead, if $X = (z, b)$, then we continue our search at node z (and we ignore the b value). When node z receives one of these "continuation searches", there are two cases: (a) The directory at z says that the key is found in a local block.

In this case we can complete the lookup. (b) The directory at z points to a third node w. In this case, the j directory was out of date, so z sends a message to j telling it to update the appropriate entry. Also, node z forwards to search to node w.

(a) Write some pseudocode that describes in more detail how lookup works. You can deviate a bit from the description above, as long as you retain the original goals (distributed, extensible hashing with lazy replication of the directory), your code works, and you explain the changes. To start you can write two top-level procedures, *lookup*, called by a client, and *continuation-lookup*. You will probably need other procedures, for example, one used by a server to tell another server that an entry is out of date.

Be sure to explain your code. You will also be graded for clarity.

(b) Explain how bucket splits (caused by the insertion of a new key) should be handled in this system. Also explain how the new bucket could be allocated in a separate server (say if the local one did not have enough space). You do not need to write pseudocode for this part; just explain in English what happens.