Database Tuning, ITU, Spring 2009

Rasmus Pagh

April 20, 2009

Project – deliverable 4

Deadline: May 5, 23.59 Danish time (extended).

Time plan for deliverable 4

April 21. Description, question/answer session.

April 28, May 5. Office hours for project supervisor (Mai Sun, in room 2A18).

May 5. Hand-in by e-mail (pdf file) to `milan@itu.dk`. Feedback meetings by appointment.

Purpose

The purpose of the practical part of this deliverable is to gain experience with concurrency control mechanisms and applying methods for tuning transactions. The purpose of the theoretical part is to train your analytical skills, based on knowledge of text indexing.

Overview

In this deliverable you continue your work on the VXL database. This time emphasis is on handling many concurrent transactions, some of which involve updates. You should use your knowledge of concurrency control, and consider the recommendations for tuning transactions given in the course.

We provide you with a program skeleton for issuing concurrent transactions to the database. You will be working with several transactions, specified below.

Transactions

19. Insert the position (x,y) of given vehicle at given time.
    public void transInsertPosition(...)

20. Find the vehicle closest to position (x,y) at given time t. For any vehicle, its position at time t is determined by the last GPS entry with timestamp not later than t.
    public int queryClosestVehicle(...)

21. Find a vehicle that can be booked between time t1 and t2, and that has a driver who is based in a given county.
    public int queryAvailableVehicles(...)

22. Book a given vehicle between time t1 and time t2, checking that it is indeed available. Unspecified fields from BookableSchedule should be set to null.
    public void transBookVehicle(...)

23. Book all available vehicles with storage capacity of at least c kilograms between time t1 and t2.
    public void transBookStrongVehicle(...)

24. Book all available vehicles with storage capacity of at least l liters between time x and y.
    public void transBookBigVehicle(...)

25. Remove from the database all information regarding a given parcel.
    public void transRemoveParcelHistory(...)

26. County c now belongs to region r. Update the database to reflect this fact.
    public void transChangeRegion(...)

In the transactions that deal with booking vehicles you should not consider vehicles that operate with a fixed schedule. The condition `vehicle_id > 16000` will suffice to select only the bookable vehicles.

Tip on locking in Oracle

A tool you may use is a mechanism for manually putting an exclusive lock on a set of rows, which will in many cases be preferable manual table locking. The SQL statement

```
SELECT * FROM R WHERE C FOR UPDATE
```

gets an exclusive lock on the rows in R satisfying condition C, and this lock is maintained until the end of the transaction. Note that you do not necessarily need to lock the row that gets updated (perhaps the transaction just adds a new row to some table) — you can use the lock to get exclusive access to some resource defined by the condition C on R, e.g., a vehicle. This of course requires that all transactions accessing the resource obtain the exclusive lock.

Theoretical investigation — text indexing

The director of the marketing department wishes to extend the VXL database with a short textual description of each customer (typically a few hundred words). The system should support free text search for a number of words (not phrases), e.g., a query for "pays big bucks" should return all customers whose description contains these three words (not necessarily in sequence). You decide not to use the DBMS's default text indexing, since the technical details are not well documented – instead, you want to implement a simple inverted list using relations, as you remember from a DBT lecture long ago. You intend to cluster the relation containing the occurrences such that Z occurrences of a word can be read in $O(Z/B)$ I/Os. A special requirement is that commonly made searches should be very effcient. Therefore you intend to keep precomputed sorted lists of documents also for pairs of words. If a query contains two words for which there is a precomputed list, this list may be used when forming the query result.

1. Consider a query for three words. With a precomputed list for two of the words, we now need to merge only two lists rather than three. However, the time for this can be much smaller than the time needed to merge the lists for each of the three words. Explain why.

2. From an execution time perspective, the best thing would be to have a precomputed list for every pair of words. Suppose the maximum number of distinct words in any document is K, and there are N words in total. Argue that the precomputed lists may be of total length around $NK/2$. (Optional: Show that $O(NK)$ is an upper bound.)

3. In view of the above, you decide to keep lists only for pairs of the 10 most common query words (45 pairs in total). What is the extra space usage needed for this, from a worst-case perspective?

4. Compare the cost of updating this index when a new document is added to the cost of updating a standard inverted index. You may assume that the ID of the new document is higher than all other IDs, and that the set of most frequent words does not change. Your answer should consider both the case where internal memory can store a block for each distinct word, and the case where the size of internal memory is very small.

To be handed in

A pdf file with the following:

- (Optional.) Description of things that were changed since the third deliverable (data model, queries, indexes).
- A description of how each transaction was implemented. This could be pure SQL, PL/SQL, or a combination of Java and SQL.
- The output of the test program.
- A description of the reasoning behind the choices made for each transaction. Were other options tried, and with what results?
- Answer to the theoretical investigation.