Assignment 1 - Database Tuning F2012

Theo Andersen - tha@itu.dk

April 2, 2012

1 Introduction

This paper describes the findings of the experiments done to fulfill the two parts of assignment 1 for the course: SDT1 - Database Tuning F2012.

Experiments were done using the *STD1_V2* Virtual box image provided by Philippe, and was executed on a Macbook pro Intel Core i5 2.4Ghz (dual core) with 4 GB ram (1Gb was given to the virtual image). The client python script was executed from the same virtual machine as the DB2 database server was running on.

This assignment focuses on comparing correctness and response time of the two isolation levels; Cursor Stability (CS) and Repeatable Read (RR). This is done by running experiments on a DB2 database with a python script doing a number of swaps and a sum on a data set of a million rows in one table. The swaps and sums run in parallel, as does the swaps individually (controlled by how many swap-threads are chosen for a particular run).

The experiments was run with 100 swaps and 10 runs.

1.1 Results

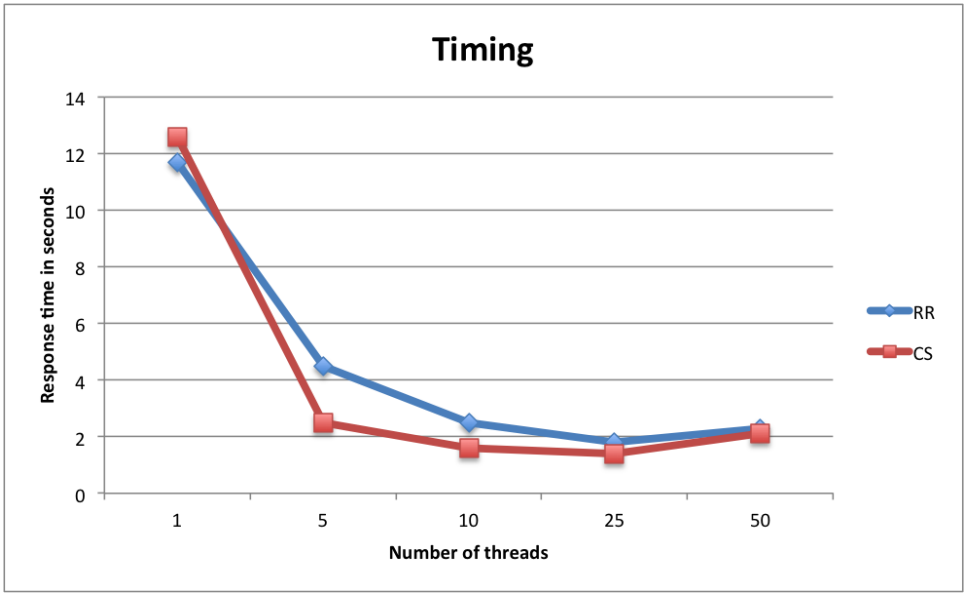
For the collection of results, the timings of each run output from the python script was used as response timings and correctness was calculated by running the sum script before running an example, and comparing this sum to the output sums of each run.

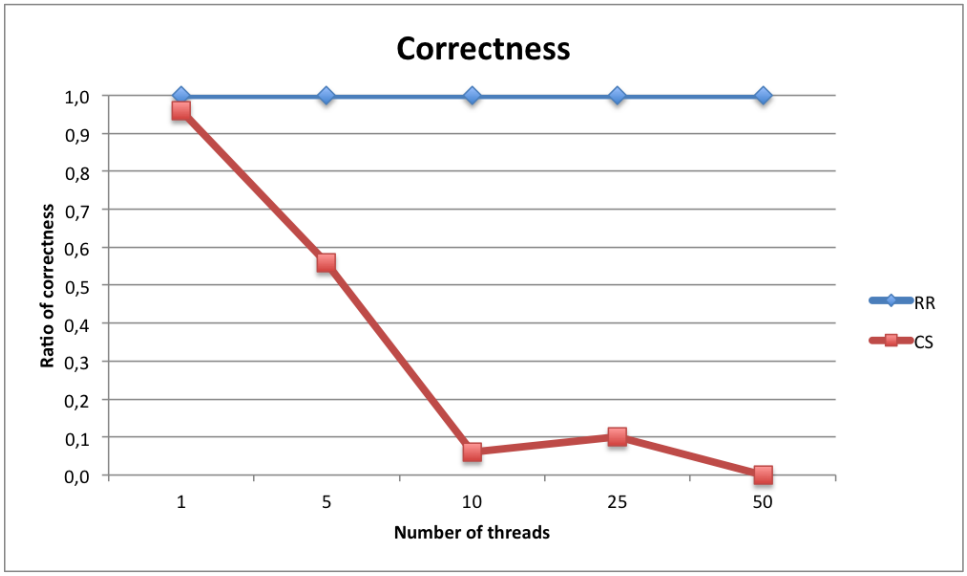
Later the python script was extended to run a separate sum on the database before creating the threads, as it was found that with many threads and heavy load on the system the correct sum could change during the example. This sum was then set to output before each run, so that for each sum there was written the previous correct sum.

2 Part 1

The first part of the assignment was to trace two graphs that represents response time and Correctness ratio for examples running on both the CS and RR -isolation levels.

Being that RR is the closest equivalent to a serializable schedule in DB2, and that CS (which is the default on DB2) is a sort of Read committed isolation level, i would think that RR would create the most correct results but be a fair bit slower than CS as there will surely be much more locking and therefore more waiting with RR.





Looking at the two graphs we can see that RR is much more correct than CS, being 100% correct in all of the examples and CS starting out with a high ratio of correctness and falling to almost nothing at 10 threads and

higher.

Both isolation levels start out at fully correct and almost the same timings. This can be accredited to the this being the results of the single-thread run, which will make each operation wait for an answer for each request to the database, and therefore from the client side assure that the server is being asked in a serializable manner.

It can also be seen that the CS seems to hit its optimal performance at around 5 threads on the test-machine, as it does not get much faster from there, but get a very low correctness ratio. This is possibly a result of the test-machine being run at its optimal on this setting, at which the last two levels of threads (25 and 50) actually seem to make the response time go a bit up again, because of the overhead of the extra threads. This might change if the client had been on a different computer that the server, as the load of running the clients, would then be kept from the database server.

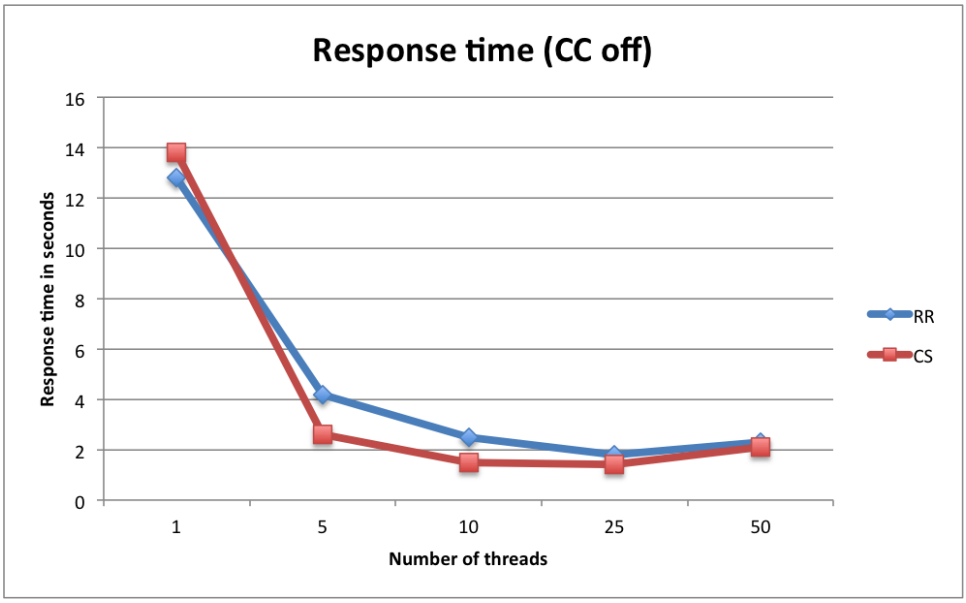
I didn't expect the CS isolation level to have quite such bad a correctness score. But it seems that the swap and sum example, really creates operations which are very dependent on atomic execution to be correct with this level of parallelism.

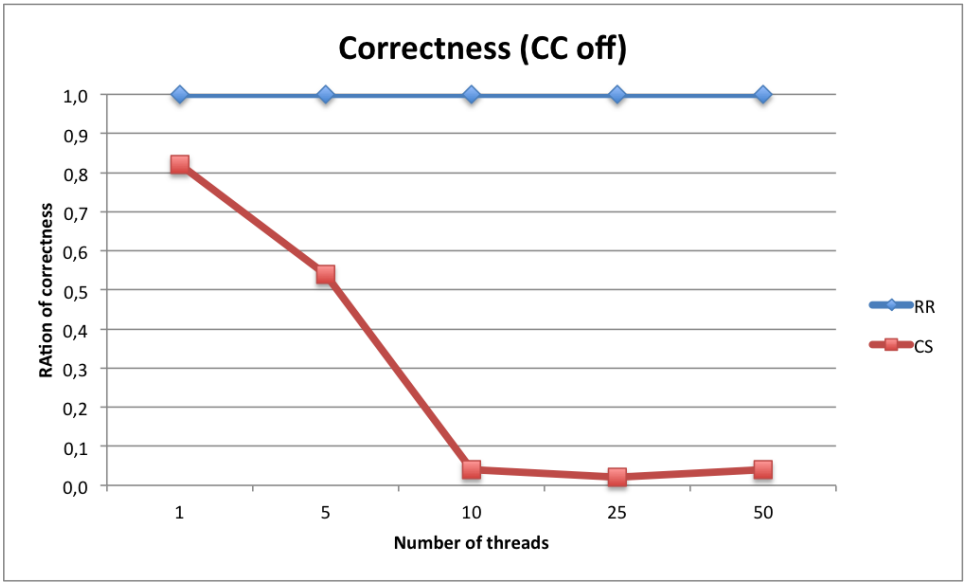
3 Part 2

This part is about the same isolation levels, now including the Currently Committed semantics.

The Currently committed semantics (CC) is a new feature for the Cursor Stability isolation level, which was introduced in DB2 v9.7. It is a feature which causes the CS isolation level to be able to read from the currently committed result instead of waiting for an update lock to be released. So instead of waiting for a lock to be released, a read can if possible read the currently committed and continue on (much like Oracles Snapshot Isolation). So the last example was actually with Currently committed semantics (CC) on, as it is default on from DB2 v.9.7.

To compare the results with this we turn CC off, reset the connections and run the examples again. Because currently committed semantics will remove some waits, i would expect these results to run faster but maybe a bit more incorrect. This because operations that now does not wait on an update, will take the last value and therefore miss an update (the sum running in parallel might miss some swaps as it will take the committed version). This will though only have an effect on the CS results, as CC semantics does not apply to RR.



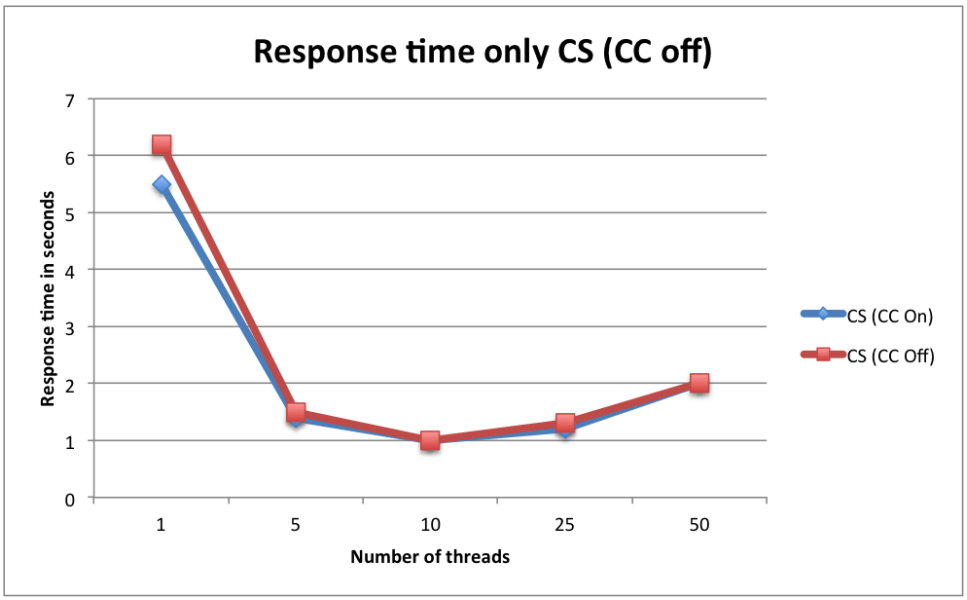


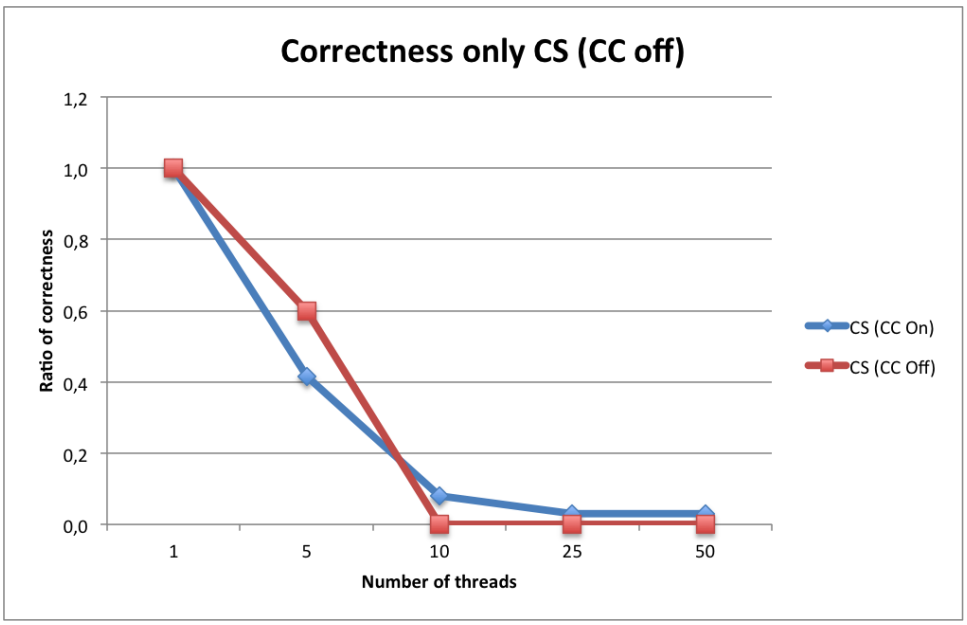
Looking at these two graphs, having CC off, we can see that the correctness is close to the previous, and the same with the timings. The RR results are the similar as the previous as expected, but the CS results are also very close to the same.

This actually confused me a bit, and i used some time on rerunning tests, confirming that tests were really being run with CC off and on. One way i tested this was that i opened two terminals, ran an update of a row in one terminal and a read of the same row in the other (with autocommit off). With CC off the read would wait not returning until the update was

committed, and with CC on the read would return the last committed right away.

With this i ran an extra number of tests only looking at the CS results, now with 50 runs instead of 10 to have even more detailed test-data, and with a lower number of swaps.





Here it is easier to see how similar the results is with CC on and off. Even with 50 runs, i don't think i can accredit the difference to anything but

test-lag. The results are very close to each other.

In the documentation it is written that CC does have an impact on the level of logging, as more logging is required to be able to read the previously committed image. This will have a slight performance penalty and might explain the similarity in performance and correctness a little.

Further more; because the example python scripts was constructed to avoid deadlocks, these examples does not show that the CC off examples would have an increased chance of deadlock.