Zhaohan Jia zj7

Yifan Li yl129

Professor Chris Jermaine

COMP 530

04 May 2017

A8 Performance Study Report

Question 1:

How did you implement the multi-threading? Was it difficult?

Did you follow my suggestions? Did you do something else?

What threading package did you use?

If you were not able to complete the assignment, then please describe what you were able to complete.

Answer 1:

First of all, we add a recursive_mutex in buffer manager. By using this mutex, we add lock/unlock pairs in the entrances/exits of the 4 functions of getPage(), getPinnedPage(). In the access() function, we add lock/unlock pairs as well.

We followed most part of the instructions in A8's pdf to implement multi-threaded implementation. In RegularSelection, ScanJoin and Aggregate, we have a field of thread numbers, and a vector to store the threads. We modified their constructors to accept an extra parameter, threadNum, that represents the number of threads we used. We transfer the run() function of them to run in separated threads. We also add mutex to lock and unlock the append() in TableReaderWriter.

To do the partition of pages in a table, we use MyDB_PageRecIteratorAlt. This iterator has a construction of low and high page to iterate, it help us to do the partition easier. But we don't use skip factor and the partition is not in an alternate way.

For ScanJoin, the first step to hash the left table's records, we don't put the this part of thread. We only do partition on the right table and make multithread there, to match the record in right table to the hashmap.

It is difficult to consider which part should be run multithread. In addition, we also need to consider which part need to be mutual exclusive for different threads, and add locks. It is also to implement aggregate into multithread.

We used the C++ 11 std libraries of mutex and thread to complete the assignment.

We implemented the multithread in BufferManager, ScanJoin, and RegularSelection. We didn't implement the thread part for aggragate.

Run each of the thirteen (non B+-Tree) RelOp tests from A6 and time the various executions using different numbers of threads. These tests are found on the following line numbers in RelOpQUnit.cc:

86, 111, 139, 163, 220, 250, 346, 385, 416, 469, 499, 550, 578

I assigned them with number 1 to 13 to record the performance and running time easier. The execution time (in seconds) of each query is listed as below:

Test case No.	Operation type	1 thread	2 threads	3 threads	4 threads
1	Regular Selection	0.005471	0.007689	0.01078	0.011352
2	Aggregation	0.000132	0.000241	7e-05	8.9e-05
3	Regular Selection	0.004381	0.005863	0.008045	0.008696
4	Aggregation	6.4e-05	5.6e-05	3.3e-05	3.7e-05
5	Scan Join	0.17273	0.212078	0.291739	0.293978
6	Aggregation	0.030467	0.030745	0.029894	0.028259
7	Aggregation	1.60517	1.61074	1.61989	1.64309

8	Aggregation	1.91065	1.91562	1.92248	1.88085
9	Aggregation	0.000296	0.000351	0.000313	0.000275
10	Scan Join	0.007816	0.088646	0.145269	0.183941
11	Scan Join	6.2e-05	5.8e-05	5.6e-05	4.5e-05
12	Scan Join	0.006213	8.3414	12.0158	6.41888
13	Aggregation	5.1e-05	33.5716	44.6442	25.012

Question 2:

Conclude your report with a paragraph or two discussion of whether or not your multi-threaded implementation was effective. If not, why was it not effective? What might you have done differently to make it better?

Answer 2:

Our implementation of multithread doesn't give us a significant speedup, however, it decrease the speed somehow. That may because, we add locks in the beginning and end of almost all the functions of buffer manager. So it may influence on the access of records. In TableReaderWriter, we use append record for output table, since our append page doesn't work well. So it may influence on the speed of writing records in the output table.

We did tried to improve out solution. In buffer manager's access() function, we don't add lock if the page was just accessed, unless we need to add a new thread pinned page. Because we have a map to keep track of the thread pinned pages, when we add a thread pinned page, the size of map would be changed, so we need to lock it to prevent multiple thread tries to modify it. In the other situation, one thread just updates their own location, it not necessarily to be mutual exclusive. Since the access() is the most frequently called function, the modification does improve the speed.