DEPARTMENT OF COMPUTER SCIENCE, HANYANG UNIVERSITY

Parallel Join Processing

Concurrent Programming

Programming Project #1

Final due date: October 10, 2021 (HARD DEADLINE)

1 TASK OVERVIEW

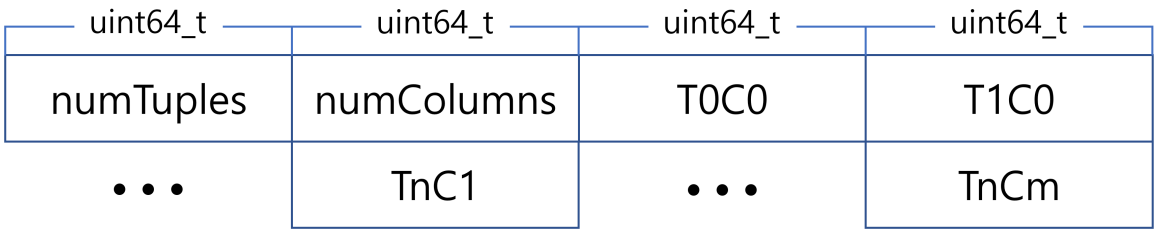
In general, relational databases store data in multiple tables. Hence, acquiring a result often involves an operation of combining data from the associated tables. In relational databases, join performs the operation of combining the rows of multiple tables based on related columns.

Your task is to evaluate batches of join queries on a set of pre-defined relations. Each join query specifies a set of relations, join predicates, and selections. The challenge is to execute the queries as fast as possible.

Inputs to your program will be provided on the standard input, and the output must appear on the standard output.

2 Task Detail

Our test harness will first feed a set of relations to your program's standard input by passing the correlated file paths. The relation files are already in a binary format and thus do not require extra parsing. Our quickstart package already contains a sample code that mmaps() the binary-formatted relations into memory. The binary format of a relation consists of a header and a data section. The header contains the number of tuples and the number of columns. The data section follows the header and stores all tuples using a column storage. Therefore, all of the values of a column are stored sequentially, followed by the values of the next column, and so on. The overall binary format is as follows (TnCm stands for tuple n of column m):



< The overall binary format of relations >

After sending the set of relations, our test harness will send a line containing the string "Done". Next, our test harness will wait for **1s** before it starts sending queries. This gives you time to prepare for the workload, e.g., sampling of the relations. The test harness sends the workload in batches: A workload batch contains a set of join queries (each line represents a query). A join query consists of three consecutive parts (separated by the pipe symbol '|'):

- **Relations** : A list of relations that will be joined. We will pass the ids of the relation here separated by spaces (' '). The relation ids are implicitly mapped to the relations by the order of which the relations were passed in the first phase. For instance, id 0 refers to the first relation.
- **Predicates** : Each predicate is separated by a '&'. We have two types of predicates: filter predicates and join predicates. Filter predicates are of the form: filter column

+ comparison type (greater '>' less '<' equal '=') + integer constant. Join predicates specify on which columns the relations should be joined. A join predicate is composed out of two relation-column pairs connected with an equality ('=') operator. Here, a relation is identified by its offset in the list of relations to be joined (i.e., we implicitly bind the first relation of a join query to the identifier 0, the second one to 1, etc.).

- **Projections**: A list of columns that are needed to compute the final check sum that we use to verify that the join was done properly. Similar to the join predicates, columns are denoted as relation-column pairs. Each selection is delimited by a space character (' ').

> **Example: "0 2 4|0.1=1.2&1.0=2.1&0.1>3000|0.0 1.1"**
> Translated to SQL:
> **SELECT** SUM("0".c0), SUM("1".c1)
> **FROM** r0 "0", r2 "1", r4 "2"
> **WHERE** 0.c1=1.c2 and 1.c0=2.c1 and 0.c1>3000

The end of a batch is indicated by a line containing the character 'F'. Our test harness will then wait for the results to be written to your program's standard output. For each join query, your program is required to output a line containing the check sums of the individual projections separated by spaces (e.g., "42 4711"). If there is no qualifying tuple, each check sum should return "NULL" like in SQL. Once the results have been received, we will start delivering the next workload batch.

For your check sums, you do not have to worry about numeric overflows as long as you are using 64 bit unsigned integers.

Your solution will be evaluated for correctness and execution time. Execution time measurement starts immediately after the 1s waiting period. You are free to fully utilize the waiting period for any kind of pre-processing.

3 TIPS

- `Logic to execute join queries // Joiner.cpp`
- `Logic to parse the input queries // Parser.cpp`
- `Logic related to operator executions // Operators.cpp`

4 TEST PROGRAM

A test program is provided to support the implementation of the assignment. The corresponding code is ***harness.cpp***, and it is built together when the program is built by executing ***compile.sh***. The test program can be executed through ***runTestharness.sh***. When ***runTestharness.sh*** is executed, the query in the **small.work** file is executed for the DBs written in the **small.init** file. Whenever the corresponding query is executed, the output result value is compared with the result value in the **small.result** file, and if there is no problem with the result, the measured execution time is returned. These files(**small.xxxx**) can be found in the directory: **workloads/small**.

The files used in the test program are written in ***runTestharness.sh***. It means that the test program selects DBs to be used for testing and execute a query on these DBs, using the information written in ***runTestharness.sh***. To test additional workload, you can also use the method of changing the contents of the ***runTestharness.sh*** file.

5 SUBMISSION

After downloading the ***submission.tar.gz*** file, unzip the file and complete the assignment. For submission, compress the source code into ***submission.tar.gz*** by executing ***package.sh*** given in the directory. Then, the created ***submission.tar.gz*** file should be submitted within a directory named: ***project*1**. In this directory, there should be only one file: ***submission.tar.gz*** (i.e. ***your_git_repo/project*1/*submission.tar.gz***).

6 Test Protocol

Our testing infrastructure will automatically evaluate the submission based on your latest gitlab commit. After unpacking the submission file, we will compile the submitted code (using your submitted ***compile.sh*** script), and then run a series of tests (using your submitted ***run.sh*** script). Each test uses the test harness to supply an initial dataset and a workload to the submitted program. The per-test time includes the time to ingest the dataset and the time to process the workload.

7 Test Environment

Submissions will be tested in a server with the following environment:

Processor	2 x Intel Xeon CPU E5-2697 v2 @ 2.70Ghz
Configuration	24 Cores / 48 Hyperthreads
Main Memory	256 GiB / no swap space
Storage	Intel SSD DC P3700 Series 1.8TB

8 Assignment Score

Scoring of this assignment will be determined by your ranking, documentation and comments. Correctness must be fulfilled as a prerequisite for ranking, and the ranks will be then decided by the running time. The last committed version should be your best one because the final grading will be processed based on your last commit before the deadline. You should submit the documentation in the form of a Powerpoint Presentation or a gitlab wiki. The documentation should include your algorithm and implementation details and other descriptions if needed. Note that the documentation should be written in a professional manner. Comments for your code is a necessity. You must include useful comments that well describes your code.

9 General Requirements

1. You should upload your code to hconnect
    - Note that email submission will not be accepted!
2. Brainstorm with your classmates on/offline.
    - Piazza can be a good dev community. Feel free to share your thoughts with your friends.
3. Do **NOT** share/use the code on public.
    - **Plagiarism issue will not be forgiven under any circumstances as mentioned before. Be careful not to upload any of your code on a public community since both the contributor and the plagiarist will be treated the same.**
4. Evaluation will be performed with the last submitted version before the deadline. Any updates after the deadline are invisible to the test program.