Software Development for Information Systems

WINTER SEMESTER 2018-2019

ATHANASIOS FILIPPIDIS 1115201400215
EVANGELOS PEKOS 1115201400157

ATHENS 2019

Project based on Sigmod 2018 for the class Software Development for Information Systems of University of Athens

This project was implemented in three different parts.
During the first part we implemented the Radix Hash Join function which has as arguments two columns of a relation and returns as a result the inner join of those two. It is called Radix Hash because the implementation of the join is splitting the data in hash buckets in order to short the data of its column. After that, while trying to join the buckets of the two columns an heuristic algorithm is used which, in short, is trying to find the join of the two columns by picking one tuple of the first and searching for the matching tuple(s) in an array of tuples of the second that have the same hash value with the tuple of the first column. This way we are trying to avoid checking as many “irrelevant” tuples and joins as possible.
In the second part we implemented a parser that in the start of the program loads all the data sets in order for them to be ready for querying. After that, we implemented a parser for the actual queries which firstly reads the relations that will be used for each query, then reads the query itself which can include either a join between two columns or a filter for a column (available filters are “=”, “>” and “<”) between that column and a number and in the end reads the columns that are to be projected. After having that information available we start to execute the queries, batch per batch. Between two batches’ execution the program sleeps for one second as we were instructed. Finally, we are printing the checksums (the sum of the tuples of each column that we were asked to project). There is also an option implemented as a define guard to print and the actual results which is currently turned off to make the program’s output more easy to be read.
In the third part we had as a goal to optimize the already existing code. Having that in mind we added threading and a job scheduler in the Radix Hash Join such as one thread is responsible for one bucket during the phase of re-ordering our columns. We also used the threads and the scheduler during the join part, again assigning one bucket to be joined to one thread each time. One more part that was optimized was the order in which the joins are being executed. We create some statistics structures while parsing our datasets and based on those are able to have a prediction of which joins should be executed first instead of others. The cost function in which we based our optimization and our Join Enumeration in this part is counting the number of “middle” results (the temporary number of tuples that are occuring from each join or filter of that query). We have noticed some significant improvement due to the Join Enumeration in the contrast of the threading optimization which seems to make the time of execution worse. However we were told that this is expected for this particular data set and queries batch on which we are testing our implementation.

<u>Structs comments:</u>
On the statistics part we used the minimum value of each column, the maximum value of each column, the number of distinct tuples of each column and the total number of tuples of each column. In order to have those statistics available we created a struct similar to the one we are storing our original data. We have an array of statistics relation with size as big as the number of

relations we have in our data sets. Each statistics relation of those has an array of statistic (statistic columns) again as big as the amount of columns this relation has. For each column we are keeping the aforementioned statistics. The formulas we used to create the approximations needed for the Join Enumeration part are originated from here.

Implemented Structs:

struct JobScheduler

This struct is the job scheduler. For the synchronization we used mutexes and condition variables. The jobs are inserted in a queue (FIFO implementation). Every thread we create executes a job. The scheduler picks the first job from the queue and executes it.

typedef struct Job

This struct represents a job. It consists of the function we want to run and a pointer to a struct with all the arguments needed for the function.

struct HistogramJobArgs

The arguments for the HistogramJob. Each job takes the two relations to which we execute the join and the indeces of the relation they are assigned to fill the parted histogram. In the end after we combine the parted histograms (HistR and HistS) we will have the total histogram

struct PartitionJobArgs

Same as HistogramJob , PartionJob takes as arguments the original relations, a pointer to the allocated reordered relation and the indeces of the original relation they are assigned to iterate over.

struct JoinJobArgs

The JoinJob finally after we create the the reordered relations, finds for a specific bucket, results.In the end what we get is a number of results equal of the number of buckets. We combine these results to get the final result.

struct tuple

A tuple consists of an ID number and the payload wich is the actual value.

struct relation

A relation consists of an array of tuples and a number which indicates the size of the relation. The statistics struct is later used for optimising the queries.

struct result
The result struct is the result of a join. It contains the rowIDs of the relations participated in the join and a number which indicates the size of result in number of ids.

struct table
The table consists of an array of relations, the number of tuples and the number of columns which is actually how many relations we have.

struct rowIDtuple
This is the result of join coupling two row ID's.

struct middle_table
The middle table constructs where we store the relations that participated in a join. For every participating relation we store the ID's resulted from a previous join, in case the same relations is joined again.

Measurements:

Running without threading or Join Enumeration:
Data loading time: 0.054885 seconds
Statistics creation time: 0.000003 seconds
Queries of small.work execution time: 3.409792 seconds
Memory Used (datasets inclusive): 3,486,175,222 bytes

Running without threading but with Join Enumeration:
Data loading time: 0.057296 seconds
Statistics creation time: 0.000004 seconds
Queries of small.work execution time: 1.221626 seconds
Memory Used (datasets inclusive): 724,478,638 bytes

Running using 1 thread and Join Enumeration:
Data loading time: 0.058663 seconds
Statistics creation time: 0.000004 seconds
Queries of small.work execution time: 1.914351 seconds
Memory Used (datasets inclusive): 895,370,420 bytes

Running using 2 threads and Join Enumeration:
Data loading time: 0.059381 seconds
Statistics creation time: 0.000008 seconds
Queries of small.work execution time: 1.921524 seconds
Memory Used (datasets inclusive): 895,400,684 bytes

Running using 4 threads and Join Enumeration:
Data loading time: 0.058048 seconds
Statistics creation time: 0.000003 seconds
Queries of small.work execution time: 1.921566 seconds
Memory Used (datasets inclusive): 895,461,212 bytes

Running using 8 threads and Join Enumeration:
Data loading time: 0.054951 seconds
Statistics creation time: 0.000004 seconds
Queries of small.work execution time: 2.084282 seconds
Memory Used (datasets inclusive): 895,793,340 bytes

Running using 8 threads but no Join Enumeration:
Data loading time: 0.059126 seconds
Statistics creation time: 0.000003 seconds
Queries of small.work execution time: 5.623615 seconds
Memory Used (datasets inclusive): 4,039,119,452 bytes

The measurements have been made in an Oracle VM VirtualBox running Ubuntu 16.04 64bit with 6554 MB Ram and 4 CPU Processors (i5 7600k 3.8GHz).

There is a make file available for compilation, just type “make” to compile. The executable file produced will be by the name “project” so execute using ./project. After that it will load the datasets searching for a file named “short.init” in which should be included the names of the relations data sets that should be loaded for later use. After that follow the instructions, press Q (as query) to execute a query and type the name of the file that includes the batch(es) or type Exit to exit the program.