Distributed Data Processing

Implementation of different join algorithms

Kostoglou Sofia 117

1. Introduction

The project aims in the implementation of two outer join algorithms: **pipelined hash join** and **semi-join**. The concept is to have a simulated distributed environment, using two different databases, in this occasion redis db, and see how the algorithms perform in such case, regarding the size of the databases, the number of same keys and the difference threshold we set for the timestamps.

2. Redis databases

The 2 databases contain tuples with one key and one value, and we consider the latter to be the timestamp.

Regarding the keys, they are random generated numbers from 0 until a limit set by the user, and according to a threshold, we can choose how many of them will be the same in the 2 databases.

Regarding the timestamps they randomly generated, considering two dates, generating various dates between them, and finally using timestamp function to get the POSIX timestamp of the random dates as a large float number, considered to be the timestamp value.

There are some parameters that the user can choose, such as db1 number of records, how many more records db2 will have, the random number limit, which is the limit number that the keys can have as they are random generated too, and the same threshold, which is a value that characterizes how many keys are common between the 2 databases.

All those parameters give us the opportunity to test the application and the performance of algorithms under various different occasions.

3. Pipelined hash join

This algorithm combines hash and pipeline techniques to efficiently perform a join. The idea is to probe and insert each value of the two databases one by one. The algorithm that can work in a streaming environment. It constructs two hash tables one for each input. For each input tuple, the input hash table is updated and the hash table of the other db is probed, to maybe get a join pair.

We get the keys and the values of each db in different lists, to use them after in the 2 functions, the **phj** which performs the join and the **probe_insert**, which is responsible for the probing and inserting phase.

First two hash tables for each db are created, and as long as there are still tuples in the dbs, we iterate over the tuples of the first db, and we insert and probe the tuple from db1 into hash table 1 and hash table 2, and then we insert and probe the tuple from db2 into hash table 2 and hash table 1.

Regarding the probe insert method, it reads the current tuple key as probe key, and it inserts the tuple in the insert hash table. If the probe key exists in the probe hash table and the timestamp

difference is less than the given threshold, the result is assigned that triple (probe key, probe table value and current value tuple).

4. Semi join

Semi Join algorithm is a case of join where the keys of the smallest db, regarding number of tuples, are retrieved and transferred to the biggest db. Then, the big db is queried using those join keys. Semi Join normally returns only the schema of left relation for the same join keys with the right relation. But here values of both key-value pairs are returned in the join result, just like in the hash join algorithm.

There are two functions, the first one is **get_keys_of_small_relation**, which finds both db sizes and returns the keys of the small one, and the function that performs the join, **semi_join**, which iterates the keys returned by the first function, gets the values of those keys for both dbs and if the timestamp difference condition is satisfied, it appends the values in the final result.

First, we get the keys of the smallest db, by measuring the db sizes, and then for each key in the small db, get the value of both dbs for that key, and after checking the timestamp difference, append the tuple in the join result.

5. Experiments

	Exp.1	Exp.2	Exp.3
#tuples in db1	100000	250000	2000
#tuples in db2	104052	251800	2100
#same keys	34310	53674	1399
Diff threshold	1000000000	100000000	10000000
Exec. time semi-join (ms)	3933	29891	250
Exec time phj (ms)	60	5384	2

Many experiments were executed, to compare the execution time of the algorithms under different parameters, as we can see in every different column in the table above. As it was expected, the more the data in the 2 dbs, the more the time consuption. We also see the phj seems to be faster than the semi-join. Also, regarding the timestamp difference threshold, the more strict it is, the less key value pairs are returned by the join algorithms.

To run the project, just cd into the project directory, open a terminal and run the command:

docker-compose up