CS392F: Automatic Software Design

P7: Gamma Joins

Jianyu Huang Xiaohui Chen

jianyu@cs.utexas.edu xhchen0328@utexas.edu

Requirement for the write-up

Explain and demonstrate that you have implemented the final parallel hash-join architecture in the lecture notes. You should have substantial tests for each component that you write. And you should have a substantial write-up of what you have done, how you have proceeded, and your organization.

Running instructions

There are 3 ways to run the test automatically.

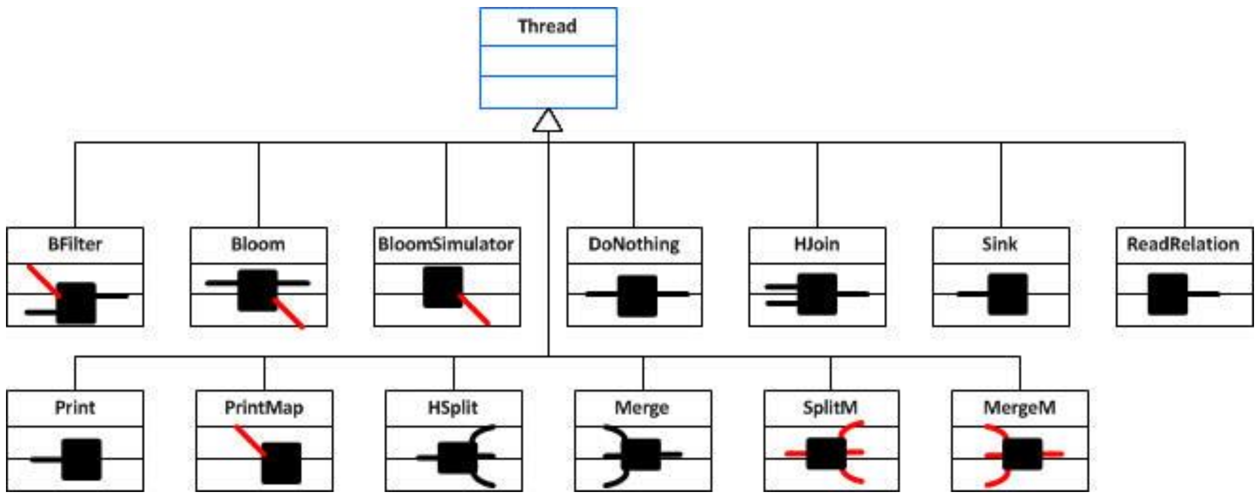
1.In ***Windows***, run "run.script.bat"(left click it, and it will run automatically)

2.With ***Cygwin***,
2.1 make the script executable by "*chmod u+x run.script.sh*"
2.2 run the script by "*bash run.script.sh"*

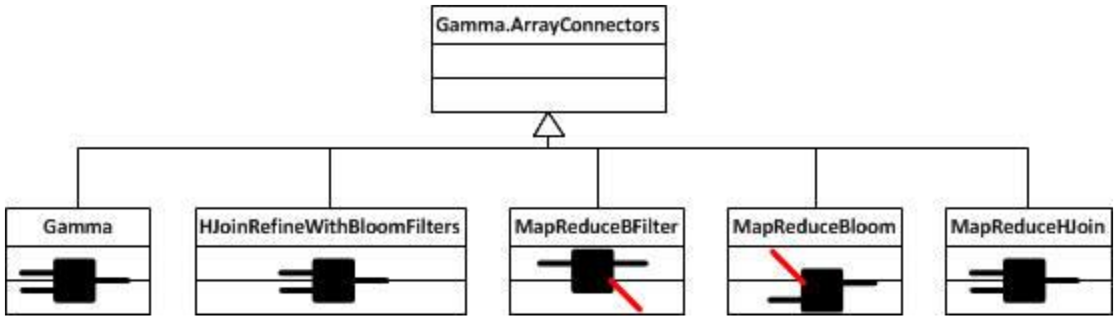
3.In ***Eclipse/NetBeans***, run "TestWrapper.java" under test folder.(You may need to add hamcrest-all-1.3.jar, junit-4.11.jar, RegTest.jar into your build path)

Code Organization

Following the step-wise approach instructions of programming assignment, our code organization looks like the followings:



On top of the above boxes, we implement pipe-and-filter graphs that implement the functionality of the primitive boxes. These boxes implement various non-primitive implementations of the HJoin box/primitive. The terminal boxes are subclasses of Gamma.ArrayConnectors, which provides useful ways of creating arrays of pipes in one call.



Implementation Details

ReadRelation

We start by implement a ReadRelation component that reads a text file and produces a stream of Strings that represent records. We decide to use regular expression and split() function to parse lines of input database files.

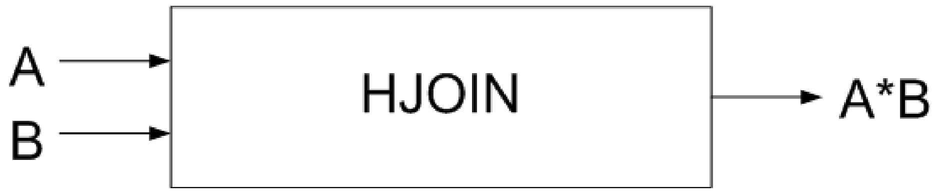
Print

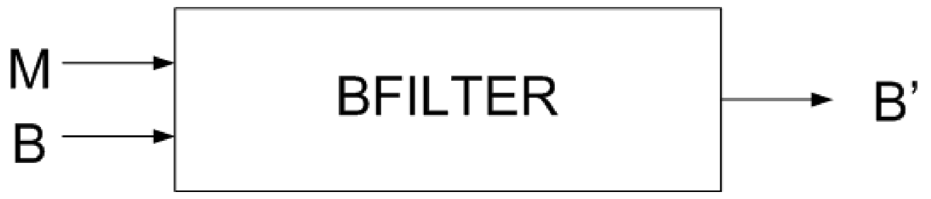
We implement a Print component that prints a stream of tuples to standard out.

HJOIN, BFILTER, etc

Hash join takes 2 streams of tuples (A,) as input and produces the join of these streams (A⋈B), BFILTER use bloom filter to filter part inputs so that the input domain for the later join operation will become smaller.

HJOIN and BFILTER both looks like a two-inputs one-output black box as the followings:



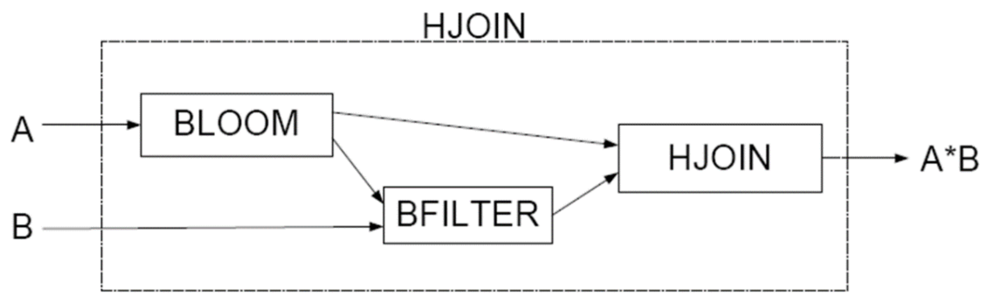


Following the instructions in the slides, we implement HJOIN and BFILTER.

For HJOIN, first we read all of stream A into a main memory hash table, then we read B stream one tuple at a time and finally hash join key of B’s tuple and join it to all A tuple’s with the same join key. We use linear algorithm in that each A,B tuples is read only once.

For BFILTER, The filtering part of Bloom filters eliminates B tuples that cannot join with A tuples. First we read bit map M, then we read each tuple of B, hash its join key: if corresponding bit in M is not set, we simply discard tuple (as it will never join with A tuples), otherwise, we output tuple.

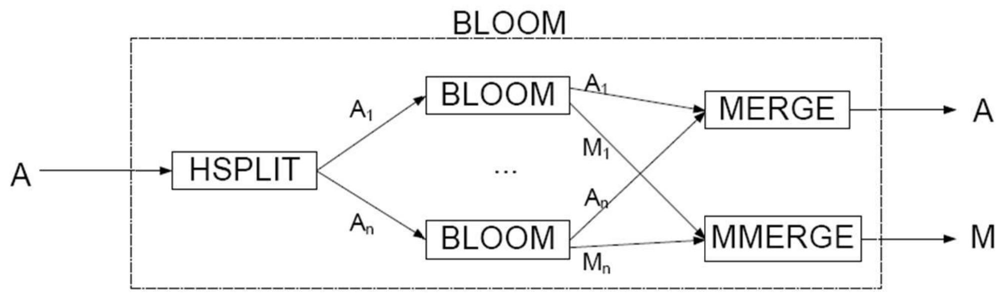
So the initial unoptimized version of gamma filter looks like the following:



Implementation of the final parallel hash-join architecture

Parallelization of BLOOM Box

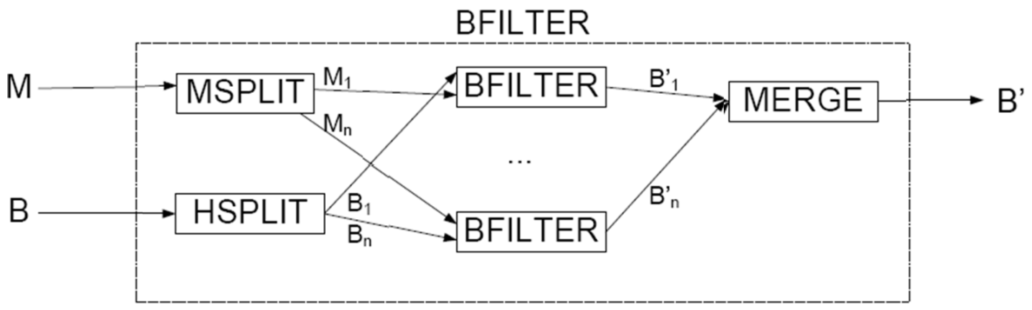
Following the instructions in the slides, we parallelize BLOOM Box as the followings:



We first HSPLIT stream A and compute Bloom filter on each substream, then reconstitute stream A by forming merge bit maps to produce single bit map M.

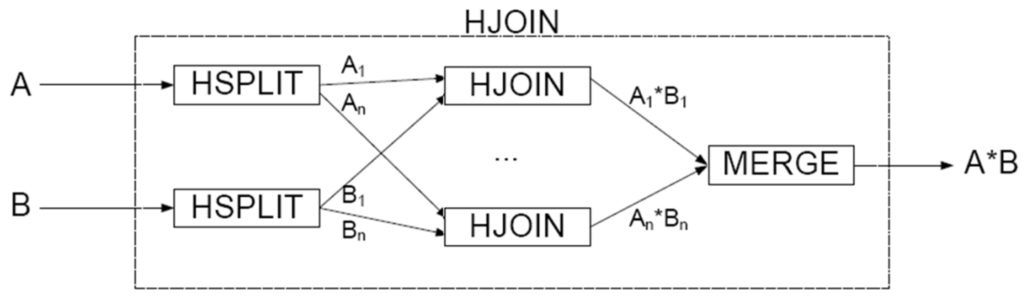
Parallelization of BFILTER Box

Following the instructions in the slides, we parallelize BFILTER Box as the followings:



We split *M* into $M_1 \ldots M_n$, and hash split stream *B* into $B_1 \ldots B_n$, then we filter B_i substreams in parallel, finally we reconstitute stream *B'*.

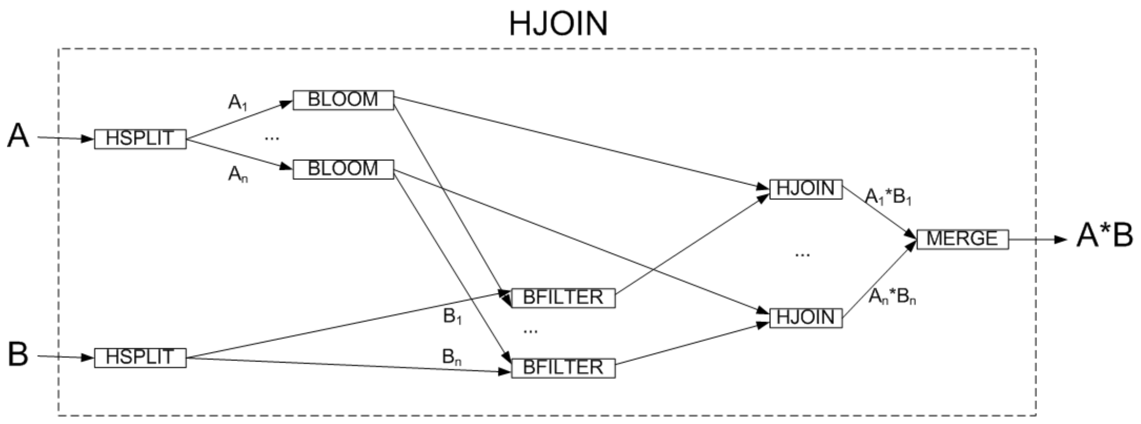
Parallelization of HJOIN Box



We split both streams using same hash function so that A and B tuples can join only if they have the same hash key, then we perform *n* joins (rather than n^2) in parallel, finally we reconstitute the join operation and merge the results.

Final Design

The final design of Gamma Join looks like the following:



MapReduce Boxes

The classes start with “MapReduce” are nothing new. They are indeed the encapsulation of several boxes. Specifically, MapReduceBoom includes the HSPlit and the 4 Bloom boxes connected to the HSplit. MapReduceFilter includes the HSplit and the BFilter boxes connected to HSplit. Finally, the MapReduceHJoin includes the four HJoin boxes and the Merge box. The HJoinRefineWithBloomFilters in the box which includes the map reduced boxes.

Testing Details

The MainTest class in hashJoins package simply calls the Gamma box in gamma package for all the test cases given. Then the gamma join results are printed out. The main testing of this project is the Regression Test

The Regression Test redirects the standard output to a txt file. Then the output would be compare to the expected results (also stored in txt format). Most of the test cases are the orders table and odetails table. For convenience they are stored in input/test/test_table and test_table1 respectively. Each regression test would be described below. All the output of the tests are stored in input/test/XXX.txt, where XXX is the class name. The expected results are stored in input/test/XXXResult.txt

TestBloomFilter

This regression test controls the input to the BFilter box. Then it examines the output.

TestBloomSimulator

The test case is test_table. The BloomSimulator output the first bitmap only. However, due to the nature of ReadEnd, the PrintMap class will output additional 3 empty lines. Therefore the output is correct as long as one of the four lines has the correct bitmap and the other 3 lines are empty lines.

TestGamma

This regression test examines the gamma join on clientXviewing, ordersXodetails and partsXodetails. The three outputs are examined by comparing with the expected joined tables

TestHSplit

This regression test controls the input to the HSplit box. Then it examines the outputs of the four HSplit boxes.

TestJoinMerge

This regression test controls the input to the four HJoin boxes and the Merge box. Then it examines the output of Merge.

TestJoinWBloom

This regression test controls the input to the HJoinRefineWithBloomFilters box. The output is indeed the hash join of the two input tables

TestMapReduceXXX

Similar to the regression tests of the components in hashJoins package, the regression tests use test_table and test_table1 to examine whether the map reduced components work as intended

TestBloomFilter

This regression test controls the input to the BFilter box. Then it examines the output.

TestReadRelation

This regression test simply use test_table as input. After the table is processed, the output is examined to check whether the output is still the same table.