Project-9

Akanksha Bansal ([akankshabansal90@gmail.com](mailto:akankshabansal90@gmail.com))

Gaurav Nanda ([gaurav324@gmail.com](mailto:gaurav324@gmail.com))

Prateek Aggarawal ([prat0318@gmail.com](mailto:prat0318@gmail.com))

* *And you should have a substantial write-up of what you have done, how you have proceeded, and your organization.*

We started by implementing ReadRelation component that reads a text file and produces a stream of tuples. Instead of using Java's **StringTokenizer,** we are using a simple regular expression to tokenize the tuples. Next we proceeded to implement **PrintTuple** class, which prints a stream of tuples to standard output.

After building a basic Connecter/Reading/Writing infrastructure, we spent some time in understanding the BMap class. It was not very intuitive in the earlier phases that we could split a BMap and still ensure the integrity of the joins. However, a re-examination of the code led us to understand that each row in BMap represents data for a separate thread. Moreover, we are using hash of Joinkeys to split data among various threads. The same criterion is used to split BMap for feeding into bloom filters. Therefore, correctness of the join is always ensured.

Then we started implementing all the components inheriting from Thread, as demonstrated in the final design discussed in the class. After developing each and every component, we wrote some sample tests cases in **“MainTest.java”**. They were primarily written by developers to debug and ensure the integrity of their component. Later on, we also added more JUnit Unit tests so that we can assert output produced by the Java program against the fixed output file.

We tested following four HashJoinRefineWithBloomFilters, HJoin, Gamma and MapReduceHashJoin boxes against all the three test cases provided in the assignment. We also demonstrate the same through the execution of the bash script.

In first version of the code, we were not using Relations class properly and were ignoring the headers present in the database files. We later introduced Relations in the final version of code. There were so many things going on initially, this approach just made the development simpler.

Once we had enough confidence in our Thread components, we moved on to implement the pipe-and-filter graphs, which implement the functionality of primitive boxes. Though these classes are supposedly inheriting from ArrayConnectors class, we did inherit them from ArrayConnectors class in our implmentation. Instead, we are using ArrayConnectors as an attribute in these classes (Composition). Still, we are not using any fancy functions of this class and just using it to represent a collection of Connectors.

Also classes like MapReduceBFilter should ideally be able to replace BFilter seamlessly. However, BFilter is a thread and MapReduceBFilter is not. But it also seemed correct to not make MapReduceBFilter as a thread, because all its components were already threads. In the final solution, we wrote “start()” method inside MapReduceBFilter, which starts all its components and exits. In this way, we can use MapReduceBFilter and BFilter interchangeably. Our test cases demonstrate this.