**Deep Bodra**

**5801 1841**

**DBI Week 8(3/23 - 3/30)**

**Detailed explanation of the topics covered in video lecture**

**Video No 32**

1. Parallelizing GLA
   1. Sequential single thread
      1. One thread executing the entire GLA
      2. No parallelization
   2. Chunk based
      1. Has one GLA per chunk
   3. Aggregation Tree
      1. Schedule ‘x’ GLA merges at a time in a hierarchical fashion
      2. This results in a tree. [binary tree for x=2]
   4. Datapath’s method using GLA Operator
      1. GLA operator handles multiple GLAs
      2. It has a queue of states
      3. Each state is managed by a CPU
      4. When a new chunk comes in
         1. Select a state if already in queue
         2. Add chunk to the state [AddItem for each item in the state]
         3. Naturally, number of states cannot be larger than the no of CPUs in the system
      5. When finished start merging
         1. Pick 2 from the queue of states and schedule them
         2. Put this state back into the queue
         3. When only 1 state remains, we are done
      6. This way the states can get very large and the merging of a large state with any other state can be very time consuming
      7. How to solve this problem? Refer previous assignments

**Video No 33**

1. External GLA
   1. We have a partitioning function which guarantees that
      1. If 2 tuples have different partition number then they don’t interact
      2. The overall result can produced by just trivivally combining the result of the different partitions
   2. In the 1st phase, partition the data and write the results out to the disk
   3. 2nd phase: Nested Loop Aggregation of GLA
      1. Idea: Build GLA for 1 partition

For t in R

If partition\_function(t) is in current partition

GLA++

else

Throw away

* 1. Using this, any GLA can be composed with any GLA using a partition function

1. Generalized Filters
   1. It abstracts the filter/select operator by having the user define a function and a state
   2. The function can be as complex as finding if a tuple is the epsilon neighborhood of some region/point or not. This introduces generalization
   3. The state must be constant so that the filter can be parallelized
   4. In case of joins the state can be a hash
   5. The filter tells whether a tuple belongs to the state or not
2. Generalized Transformers
   1. They are like Filters(i.e. They accept a function and a state) but can produce a set of tuples for each tuple
   2. For eg, Join can be implemented using these
3. Map Reduce
   1. Map: Takes a list of objects and a function to produce another list of objects by applying the function to each object in the original list
   2. Reduce: Takes a list of objects and a function to produce a single object (Just like aggregation)

**Video No 34**

* 1. Data structure for implementation
     1. Strings
        1. Strings are the most simple data structures to represent any type of data
        2. But there is an overhead on conversion and also cannot be parallelized for map reduce
     2. Key value pairs
        1. Map takes a key value pair and generates a bunch of key value pairs using the user defined map function
        2. Reduce takes a key and a bunch of key value pairs and generates a key value pair the user defined reduce function
  2. Implementation
     1. Random parallelization
        1. Each node runs at least one map and a reduce task
        2. Map
           1. Generate random keys for mapper and apply the function to the value only.
           2. Output key will remain same as the input key
        3. Combine
           1. It reduces the key value pairs locally
           2. Each node is assigned a particular key value on which it will perform a reduce job
           3. The nodes send the required key value pairs to the designated nodes
        4. Reduce
           1. All key value pairs with same keys will go to the same reduce task
           2. The reducer applies the reduce function on the value.
           3. It may or may not change the key
        5. Drawback
           1. Quote from the lecture “Combiner is like a reducer but applied locally on the top of what map does. But that is where the trouble starts. It really feels bad. As an abstraction it feels weird. If it is like a reducer then why is not a reducer”
           2. Compare this to GLA, AddState can be used locally as well as globally [So much better and consistent]

--THANK YOU--