**Deep Bodra**

**5801 1841**

**DBI Week 7(2/17 - 2/21)**

**Detailed explanation of the topics covered in class**

**2/18/2020**

1. Index Based Algorithms
   1. Efficient for transactional queries
   2. They are better for unique valued attributes
   3. We can have index on multiple attributes (say A,B), but the queries with A will only be accelerated as the order in which we specify index matters
   4. Index Based Join

For t in R

U = Index.Lookup(t.A)

For u in U

Output t.u

* 1. This is still slower than linear scan because of the random reads
  2. It works at the rate of just 20 tuples/second

1. Clustering vs Secondary Indexing
   1. Clustering uses data structures like B+ trees
   2. Secondary indexing is based on having external pointers like (Disk block number, tuple number)
2. B+ trees
   1. The records are stored only at the leaf nodes and in sorted order
   2. The internal nodes contain the keys and direct us towards the leaf node we are interested in
   3. The leaf nodes are connected in the form of a linked list, so that we can traverse the records without having to go back up the tree
   4. Just like a binary search tree, we either go in a left subtree or the right sub tree of a key (out of many keys at a node) based on <= or > comparison
   5. Drawback
      1. Slow
         1. Linear scan beats B+trees
         2. Solution: split a node if the tree exceeds a threshold number of keys
      2. Cannot be parallelized
   6. Balancing tuples can be avoided if we have very fast external sort and if we build the tree bottom up
3. Datapath
   1. Remember? We have chunks in datapath
   2. We keep some meta data for each chunk
      1. Min and max values of attributes
   3. So we read a chunk only if the target values lies in this range
4. Hash Index (External)
   1. The hash function produces a number in the range [0,B-1] where B is the number of buckets
   2. If a bucket fills up then we add a new bucket and link the old and the new bucket by maintaining a pointer
   3. We need a true random number generator for this
   4. Analogy: N balls and M bins
5. Implementation of operators
   1. Duplicate elimination
      1. H.Init()

For t in R

If !H.find(t)

H.insert(t)

output(t)

* + 1. H.size is proportional to no of distinct tuples
    2. This is a non blocking algorithm
  1. Group by (with aggregate functions)
     1. A,B=>group by attributes
     2. Algo

H.hash(key(A, B))

H.Init()

For t in R

group=H.find(key(t.A, t.B))

If !group

group=new group(A,B,...)

H.Insert(group)

group.Aggregatei.AddItem(t) for i=0,1,2...n

For group in hash

output(g.A, g.B, Aggregatei.Finalize()) for i=0,1,..n

* + 1. This is a blocking algorithm i.e. nothing happens and then everything happens

RULE OF THUMB: All operators except CROSS PRODUCT benefit from hashing some way or the other

* 1. R Join S

QUICK TIP: S should be smaller of the two relations and Try projecting S for better efficiency

Algo

Hash(S)

Scan(R)

For u in S

H.Insert(u)

For t in R

U = H.lookup(t.A)

For u in U

Output t.u

* 1. SnowFlake Schema
     1. A relation is broken down into multiple tables such that the fact table contains the primary key and they are mapped to the foreign keys of the dimension tables

**2/20/2020**

1. Implementation of Operators
   1. Join
      1. Blocking Join

….

/ ...

Join2

/ \

Join1 S2

/ \

R S1

* + - 1. R=fact table
      2. Si = dimension table (comparatively smaller)
      3. We wait for Si to get hashed completely
      4. Thus blocking
      5. Despite that it is faster than non blocking(discussed below)
    1. Non blocking

/ \

/ \

HR HS

|t |u

| |

R S

Let, t belongs to R and u belongs to S

Algo

If t in Hs

Combine t and tuples of Hs

Else

Add t to HR

Similarly for u

* + - 1. Intuition behind correctness of this method
         1. Every tuple that should be a part of the join is produced by this tuple
         2. Every tuple produced by this is a part of the join
      2. It is a nightmare to parallelize this because we have 2 hashes and there will be write conflicts in a multithreaded environment
  1. R ⋂ S
     1. It is similar to above
     2. Trick for duplicate elimination:

If t in Hs

Remove from Hs

* 1. R\S
     1. Trivial
  2. R U S

Think at home

--THANK YOU--