## Chapter 4.8 Using Algorithm more than Twice - Pass

Through popularizing the algorithm, then choose to use more pass to deal with the random size Relation. In this Chapter, we only consider the method based on Sort and Hash.

### Chapter 4.8.1 Multi - Pass Algorithm based on Sorting

***Prerequisite:***

Here we extend *TPMMS* to *Three Passes Algorithm*. Actually, we have the Simple Recursive Sort Algorithm. No matter how big the Relation is, it can be sorted sequentially. Or if we are willing to, for random value n, we can build n sorted sequences.

***Assumption:***

Here we have M available main memory blocks to sort Relation R, and the Relation R is stored according to cluster. Then, doing as follow:

* *Basics:*

If Relation R can be stored into M blocks *(B(R) <= M)*, then just read Relation R into the main memory, and sort the Relation according to your favorite Sorting Algorithm, then write it back to disk.

* *Conclusion:*

If Relation R can not be stored into main memory, then divide the blocks of Relation R into M blocks, named as R1, R2, ... , RM. For each i = 1, 2, ... , M, then sort Ri recursively, then merge M sorted sequenced sub - tables.

If we not just do the Sort Operation, but do some unary operations on Relation R, such as Grouping or Selection. Then we modify the algorithm, and in the last step of Merge, just operate on the front of these sorted sequenced sub - tables.

1. *For Unary Operators:*
2. For Selection Operator, output one copy of each different tuple, and skip the other copy of this tuple.
3. For Grouping Operator, only sort on Grouping properties, and combine by using the given value on those Grouping properties.
4. *For Binary Operators:*
5. When we want to operate the Relations with Binary Operators, such as Intersection or Join, then basically using the same thinking, the only difference is divide the two Relations into the sub - tables with the M total number. Then using the recursive algorithm to sort for each sub - tables. At last, using the algorithm in the Chapters before to operate.

We assign M buffer blocks for Relation R and S. But in order to make the total pass in the least number, we normally divide all buffer areas according to the number of blocks of Relation R and S. Then divide the buffer area by proportional. The number of buffer blocks that for Relation R equals to *M( B(R) / (B(R) + B(S) ) )*, then Relation S gets the remaining number of blocks.

### Chapter 4.8.2 Performance of Multi - Pass Algorithm based on Sorting

***Introduction:***

Now, let’s consider the number of disks and the size of Relation that being operated and the relationship between the main memory.

***Calculation:***

Here, S(M, k) is the most biggest Relation that sort by using M buffer blocks and k pass.

*Basis:*

If here k equals to 1, means here just enable one pass, then here we have B(R) <= M, which means S(M, k) = M.

*Assumption:*

Here assume that k > 1, then we can divide Relation R into M pieces, each piece must be sorted by k - 1 pass. If B(R) = S(M, k), then each size of M piece of Relation R equals to S(M, k) / m, here it can not exceed S(M, k - 1), just means S(M, k) = M \* S(M, k - 1).

Then expand the recursive statement, we find that:

S(M, k) = M \* S(M, k - 1) = M ^ (k - 1) \* S(M, 1)

Since S(M - 1) = M, then we can get that M ^ (k - 1) \* M = M ^ K. Which means that B(R) <= M ^ K, then through k pass, we can sort the Relation R. Put in another words, if we want to sort Relation R in k pass, then the least number of usable buffer block equals to M = (B(R))^1/k.

The Sorting Algorithm needs to read from the disk and write back to the disk in each pass, therefore one k pass Sorting Algorithm needs 2 \* k \* B(R) times disk I/O.

***Cost:***

Now, let’s consider make the multi - join R(X, Y) with S(Y, Z) as our representation, and to consider it’s cost.

*Assumption:*

Here we assume that *j(M, k)* is k pass, then the most biggest blocks number of M buffer block is the total of blocks that the Relation we can connect. Then the total block is just equal or less than j(M, k), which is to say that *B(R) + B(S) <= j(M, k)*, then the Join Operation can realize.

In the last pass, we merge the M sub - tables of two Relation. Each sub - tables is using the k - 1 pass sorting, so each size of them will not exceed *S(M, k - 1) = M ^ (k - 1)*, then the total size is *M \* S(M, K - 1) = M ^ k*, so B(R) + B(S) <= M ^ k, then the number of blocks that we need for k pass join equals to *(B(R) + B(S)) ^ 1 / k* buffer blocks.

In order to calculate the Disk I/O for calculating the multi - algorithm, then we need to keep in mind, we do not take the join or other Relation Operation writing back to the disk into consideration. So here we are using *2 \* (B(R) + B(S)) ^ 1 / k* buffer blocks to sort the sub - tables, also *B(R) + B(S) disk I/O* to read sorted sequence. So at last, the total disk I/O equals to *(2 \* k - 1) \* (B(R) + B(S))*.

### Chapter 4.8.3 Multi - Pass Algorithm based on Hash

***Introduction***

For the operation of big Relation, there exists one method to use hash method recursively. Here we hash one or two Relations into M - 1 buckets. (M is the available main memory buckets number.)

For unary operator, we just operate each operation on each bucket.

For binary operator, such as join, we just apply each couple of operation on the corresponding buckets, just like the whole Relation.

***Basics:***

For unary operator, if the Relation can be put into M buffer blocks, then read it into memory and execute the operation. For binary operator, if one Relation can be put into M - 1 buffer blocks, then put the Relation into the main memory, then put the second Relation into Mth buffer block just one by one.

***Conclusion:***

If there has no one relation can be put into the main memory, then just hash each Relation into M - 1 buckets. Then operate on the bucket recursively on each bucket, also we accumulate output the result for each bucket.

### Chapter 4.8.4 Performance of Multi - Pass Algorithm based on Hash

***Introduction:***

When we hash a Relation into the buckets, then we want to guarantee that all tuples are divided evenly to the buckets. If we choose a real random hash function, it may nearly satisfy this assumption, but when we distribute all tuples into the bucket, then they must be uneven.

***Unary Operator:***

*Assumption:*

For example, when we are using M buffer blocks for Grouping or Selection on Relation R. Assume that u(M, k) is the biggest relation blocks that k - Trip Hash Algorithm can process. Here we can define u according to:

*Basis:*

u(M, 1) = M, because Relation R has to be stored into M buffer blocks, which is to say, B(R) <= M.

*Conclusion:*

Assume that in the first step, we divide the Relation R into M - 1 same size buckets. Then, we can calculate u(M, k - 1) according to the method below. The bucket that prepares for the next trip need to be small enough, then they can be process in the k - 1 trip, which is to say, the size of bucket is u(M, k - 1). Since the Relation R is divided into M - 1 buckets, then u(M, k) = (M - 1) \* u(M, k - 1).

After that, we can unfold the equation and find that u(M, k) = M \* (M - 1) ^ (k - 1). Also, we can assume that M is big, then u(M, k) = M ^ k, to put in another word, if M <= ( B(R) ) ^ 1 / k, we can use M buffer blocks to execute the unary operator on Relation R.

***Binary Operator:***

Here we can do a simple analysis for Binary Operator. Let’s consider the join operation. Let j(M, k) be the upper limit of two Relations R and S, operate R(X, Y) Join S(Y, Z).

*Assumption:*

M is the number of available buffer blocks, and k is the trip that we use.

*Basis:*

j(M, 1) = M - 1, then we use one trip algorithm to join, then Relation R or Relation S can be stored into M - 1 blocks.

*Conclusion:*

j(M, 1) = (M - 1) \* j(M, k - 1), which means in the first trip of k - trip traverse, we can divide each Relation into M - 1 buckets, we hope that each bucket is 1 / (M - 1) of the whole Relation, but we should connect each pair of bucket in the (k - 1)th trip.

By unfolding the cycle j(M, k), then we can conclude that j(M, k) = (M - 1) ^ k. Also assume that M is big enough and j(M, k) = M ^ k, which is to say that min( B(R), B(S) ) <= M ^ k, so we can by using k trip and M buffer blocks to realize R(X, Y) join S(Y, Z).