Concordia University
Engineering and Computer Science

Concordia University

Dept. Of Computer Science & Software Engineering

COMP 6521: Advanced Database Technology & Applications

Winter 2018

Mini Project Report

Project Name	Mini Project 1
Description	Implementation of sort-based Bag Difference
Team Members	Arun Varghese (40058590) Manohar Gunturu (40070762) Mehakpreet Singh (40057337) Kasyap Vedantam (40042347)
Date of Submission	19Th Feb 2018

Procedure/Steps followed to implement project:	1.Calculations as per main memory size constraints 2.Discussion on implementing the TPMMS method to sort the input relations 3.Design Pseudo-code to implement Bag Difference 4.Refactor code to decrease processing time and I/O's count

Modules in Project	1.Sublist Creation (Main_String.java) – Creates Sub list 2.Merge Sort (MergeSub.java) - Performs Merge sort 3. Bag Difference (BagDiff.java) – Reads data from Relations T1 & T2 and computes bag difference and results are captured in output file

Steps to run Project:	Run the "Main_String.java" on bag1 and then bag 2 by changing the file location . Run "BagDiff.java" to compute the bag difference. The results are stored in "file-output.txt".

Calculations:

I tuple = 100 bytes

Main memory Constraints - 5MB and 10 MB

For 5MB, 5*1024*1024 Bytes = 5242880 Bytes

For 10MB, 5242880 Bytes*2 = 10485760 Bytes.

Statistics on 5400 RPM 2.5" TOSHIBA SATA HDD on 5MB HEAP:

For bag 1 with 1200472 tuples:

- Our system is taking 30,000 tuples in each fill, and thereby creating 40 sublists in **5.3 sec**
- And it is doing external merge sort on 40 sublists in **5.9 seconds**

Above process repeats for sorting bag 2:

- And then we are computing bag difference on bag1 and bag2 which takes **10.3seconds**

Statistics on Intel NVM SSD on 5MB Heap:

For bag 1 with 1200472 tuples:

- Our system is taking 30,000 tuples in each fill, and thereby creating 40 sublists in **3 sec**
- And it is doing external merge sort on 40 sublists in **3.4 seconds**

Above process repeats for sorting bag 2:

- And then we are computing bag difference on bag1 and bag2 which takes **8 seconds**

To do sort on each bag, a HDD is taking 11.2 seconds and on SSD 6.4 sec, that is why organizations use SSD's for challenging workloads.

Now to do sort on two bags is 11.2*2 = 22.4, 22.4+10.3 = 32.7sec to do both sort and bag difference on two bags.

Statistics on 5400 RPM 2.5" TOSHIBA SATA HDD on 10MB HEAP:

For bag 1 with 1200472 tuples:

1. Our system is taking 40,000 tuples in each fill, and thereby creating 40 sublists in 4.4 sec.
2. And it is doing external merge sort on 40 sublists in 3.8 seconds

Above process repeats for sorting bag 2:

3. And then we are computing bag difference on bag1 and bag2 which takes 8.9seconds.

The time difference is because of the buffer size varies in 10MB case.

Calculating the IO's:

For each block move we have taken it as one IO. so a block holds 40 tuples so for 1200472/40 = 30,012 blocks. So

Read the file + write sublists + read sublists + write sorted file = 4 * 30,012 IO's.

To do it on 2 bags = 2 * 4 * 30,012.

If we add Bag diff IO's = 2 * 2 * 4 * 30,012.

How Sorting Algo effects when sorting in Phase 1:

We all know that Quick sort has O(n log n) complexity, but if recursion depth is too much then this complexity goes to $O(n^2)$.
We are using Introsort in java where it begins with quicksort and switches to heapsort when the recursion depth exceeds a level.

Important Observation we made:

- As we are running on low heap memory , it is better to not to use String classes in Java, as it uses to much of heap, we have made byte level coding as much as possible.
- Do not reach the 95% limit of heap memory, because java Garbage collection starts running, which makes program too slow and GC errors.
- As we are having low RAM memory and good process speed, we have tried to use every single tick of process instead of creating temporary objects

For example in order to convert a String to Integer, Java gives us Integer.parseInt(new String()), instead of doing that we have read it as bytes and converted bytes to Integer by doing simple mathematical calculations.

How to increase data access speed:

Even though we have studied about arranging data on consecutive cylinders and disk mirroring which increases the data read speed, those techniques needs system level programming at OS support. Of course that is one of the reason why DB2 works faster on IBM machines.

The one thing we thought that is achievable is double buffering, where we load memory into two buffers for each sublists and when one buffer is emptied we initiate IO on another thread to fill the 1st buffer, and in the mean time we continue to read 2nd buffer, for this we need two buffers for each sub-list, but as we are having less memory dividing it into small bufferes and filling them up need more IO's, and we have also faced the Garbage collection errors as Async IO's needs threads and each thread has its own stack.

So we tried to use huge buffers for both reads and writes in order to make less IO's practically and speed.

Theoretical:

	5 MB	10 MB
Bytes	5242880	10485760
No of tuples for 1 fill	52428	104857
Input tuples in each file	1200472	1200472

Practical:

	5 MB	10 MB
Bytes	5242880	10485760
No of tuples for 1 fill	29,500	40,000
Total Time (Best Case)	**24 sec**	**14 sec**
IO's	2 * 2 * 4 * 30,012	2 * 2 * 4 * 30,012

Bag Difference - Test Scenarios& Results:

After sorting, Bag difference will be performed and stored in the output file as said above.

Providing record examples with single digit values here for easy understanding, howeverrecord value is 8 digits in the program.

Loading 4 records at a time in the below examplefor computing bag difference (considering memory constraint) In actual program limit value is much higher

Providing few test scenarios below.

Test Scenario 1:(O/p Format – Bag difference count * Tuple)

T1 File	T2 File	Output
1	1	0 * 1
2	2	0 * 2
3	3	0 * 3
4	4	0 * 4

Test Scenario 2:(O/p Format – Bag difference count * Tuple)

T1 File	T2 File	Output
4	1	0 * 4
5	2	0 * 5
6	3	0 * 6
8	4	1 * 8
9	5	1 * 9
10	6	1 * 10
	7	

Test Scenario 3:(O/p Format – Bag difference count * Tuple)

T1 File	T2 File	Output
1	1	0 * 1
2	2	0 * 2
3	3	0 * 3
4	3	1 * 4
4	3	0 * 5

5	4	2 * 6
6	6	
6	6	
6	6	
6	7	
6	7	
	8	

Test Scenario 4:(O/p Format – Bag difference count * Tuple)

T1 File	T2 File	Output
1	4	1 * 1
2	5	2 * 2
2	6	1 * 3
3	7	4 * 4
4	8	1 * 5
5	9	

Test Scenario 5:(O/p Format – Bag difference count * Tuple)

T1 File	T2 File	Output
1	2	1 * 1
2	2	3 * 2
2	3	0 * 3
2	4	0 * 4
2	5	
2	6	
3	7	
4	8	

Good Luck