

**COMP 6521**

**Two-Pass Multiway Merge Sort**

Project Report

Submitted To

**Dr. N. Shiri**

|  |  |
| --- | --- |
| **Student Name** | **Student Id** |
|  |  |
| Akhila Chilukuri | 40071241 |
|  |  |
| Alekya Karicherla | 40059347 |
|  |  |
| Ezhilmani Aakaash | 40071067 |
|  |  |
| Alla Sai Krishna | 40069746 |
|  |  |

**Abstract**

Two-pass Multiway Merge sorting(TPMMS) is an external sorting algorithm that can handle massive amounts of data. It is required when the data being sorted do not fit in to main memory and used for performing relational algebra operations on relations that one-pass algorithms cannot handle, we can perform operations in two passes for very large relations. It has restrictions on the access, depending on the external storage medium. It is used to reduce the time required to sort the entire relation by using merge sort.

In tpmms we read data from relations and process, written back to disk, then re-read the data from the disk to complete the operation. It has two phases, in phase-one we repeatedly fill buffers with new tuples and sort using main memory algorithm. In sorting phase, chunks of data small enough to fit in main memory are read, sorted, and write back the sorted sublist in to secondary storage. In phase-two, merge the sorted sublists and write out to disk.

The number of disk IO’s required in phase-one is two times number of blocks because we read entire relation in terms of blocks and write out each sorted sublist on to disk in terms of blocks. In phase-two we need to re-read each of the sorted sublists again (i.e.) Equal to number of blocks and if we need to store the result on the disk it will be four times the number of blocks. In this TPMMS implementation we keep track of the duplicate records from all the sublists and calculate the bag difference which is the resultant of duplicate records from file-one and duplicate records from file-two.

**TABLE OF CONTENTS**

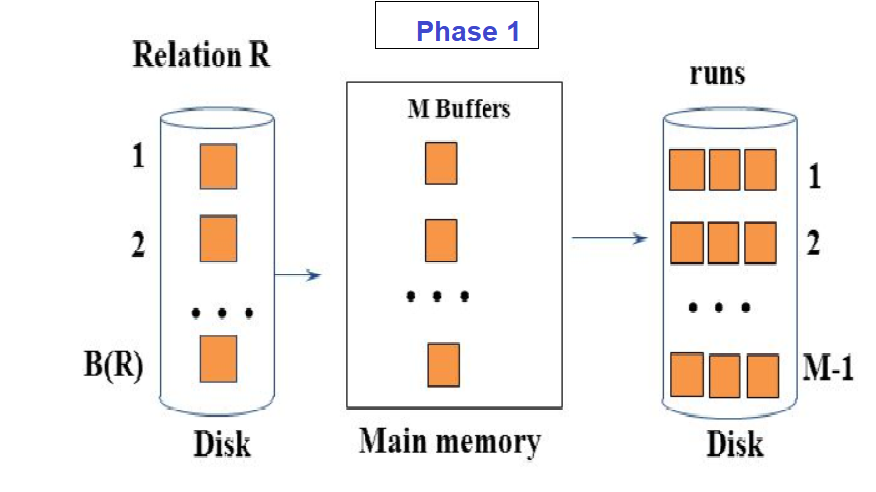
|  |  |
| --- | --- |
| [**Introductio**](#page4)**n** | **4** |
| [**Parsing and Safety Condition**](#page4)**s** | **4** |
| [Safety Condition](#page9)s | 9 |
| [**Data log: Semantics And Evaluation Paradigm**](#page10)**s** | [**1**](#page10)**0** |
| [Naive Evaluation Techniqu](#page10)e | [1](#page10)0 |
| [Semi-naive Evaluation Techniqu](#page11)e | [1](#page11)1 |
| [Bookkeepin](#page11)g | [1](#page11)2 |
| [Extension: Built In predicat](#page11)e | [1](#page11)2 |
| [**Query Evaluatio**](#page12)**n** | [**1**](#page12)**2** |
| [**Performance Evaluatio**](#page12)**n** | [**1**](#page12)**2** |
| [**Challenges Face**](#page14)**d** | [**1**](#page14)**5** |
| [**Conclusion & Future Wor**](#page14)**k** | [**1**](#page14)**6** |
| [**Tools Use**](#page15)**d** | [**1**](#page15)**6** |
| [**Work Contributio**](#page15)**n** | [**1**](#page15)**6** |
| [**Reference**](#page16)**s** | [**1**](#page16)**7** |

**1. Introduction**

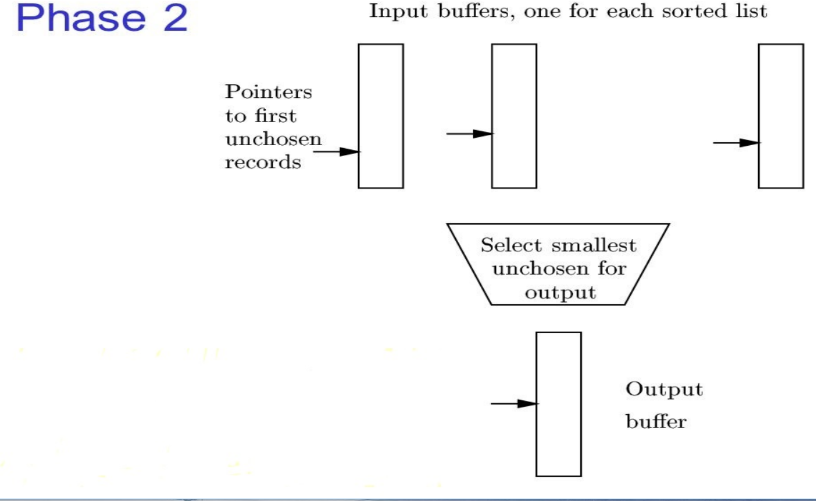
In two-pass Multiway Merge sorting(TPMMS), we process very large relations in two phases. Suppose if we have ‘M’ Bytes of Main memory, ‘N’ tuples in Relation of ‘R’ bytes each and block size is of ‘B’ Bytes. Relation size will be ‘N\*R’ Bytes and number of main memory buffers is ‘M/B’ blocks. In phase-one, fill ‘M’ buffers with new tuples from R, sort and write back to disk. After phase-one, we have ‘NR/M’ runs (sorted sublists). In phase-two, Merge sort sublists using (M/B-1) Input Buffers, one output buffers and sort the sublists into a single sorted list. In output buffer it stores the smallest element into the first available position of output block from the sorted sublists by performing search until output buffer is full, if its full write it to disk and reinitialize the output buffer to hold the next output block until all the sublists are empty and thus it will merge all the sublists into a single list. The complexity is O (N log(N/M)).

**Characteristics of two-pass multiway merge sort:**

* It processes the vast files that are unable to fit into main memory.
* Reduces the cost for input output operations.
* Minimize the number of times data moved between external storage and main memory.



In phase-one, fill ‘M’ buffers with new tuples from R, sort and write back sorted sublists to disk.



In phase-two, Merge sorted sublists using M-1 Input Buffers and one output buffers. We will sort the sublists into a single sorted list.

**Example of Two-pass multi merge sort:**

Suppose, if we haveN = 15 records, M = 3 (15 records on file T1, memory capacity: 3 records)

Ta1: 17, 3, 29, 56, 24, 18, 4, 9, 10, 6, 45, 36, 11, 43,47

Sorting of runs:

Read 3 records in main memory, sort them and store them in file t1:

17, 3, 29 -> 3, 17, 29

t1: 3, 17, 29

Read the next 3 records in main memory, sort them and store them in file t2

56, 24, 18 -> 18, 24, 56

t2: 18, 24, 56

Read the next 3 records in main memory, sort them and store them in file t1

4, 9, 10 -> 4, 9, 10

t1: 3, 17, 29, 4, 9, 10

Read the next 3 records in main memory, sort them and store them on t2

6, 45, 36 -> 6, 36, 45

t2: 18, 24, 56, 6, 36, 45

Read the next 3 records in main memory, sort them and store them on t1

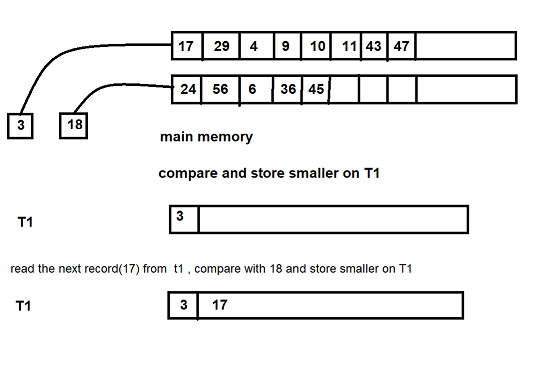
11, 43,47 -> 11, 47

t1: 3, 17, 29, 4, 9, 10, 11, 43,47

At the end of this process we will have three runs on t1 and two runs on t2:

t1: 3, 17, 29 | 4, 9, 10 | 11, 43,47

t2: 18, 24, 56 | 6, 36, 45 |



After merging the runs, we get

3,4,6,9,10,11,17,18,24,29,36,43,45,47,56.