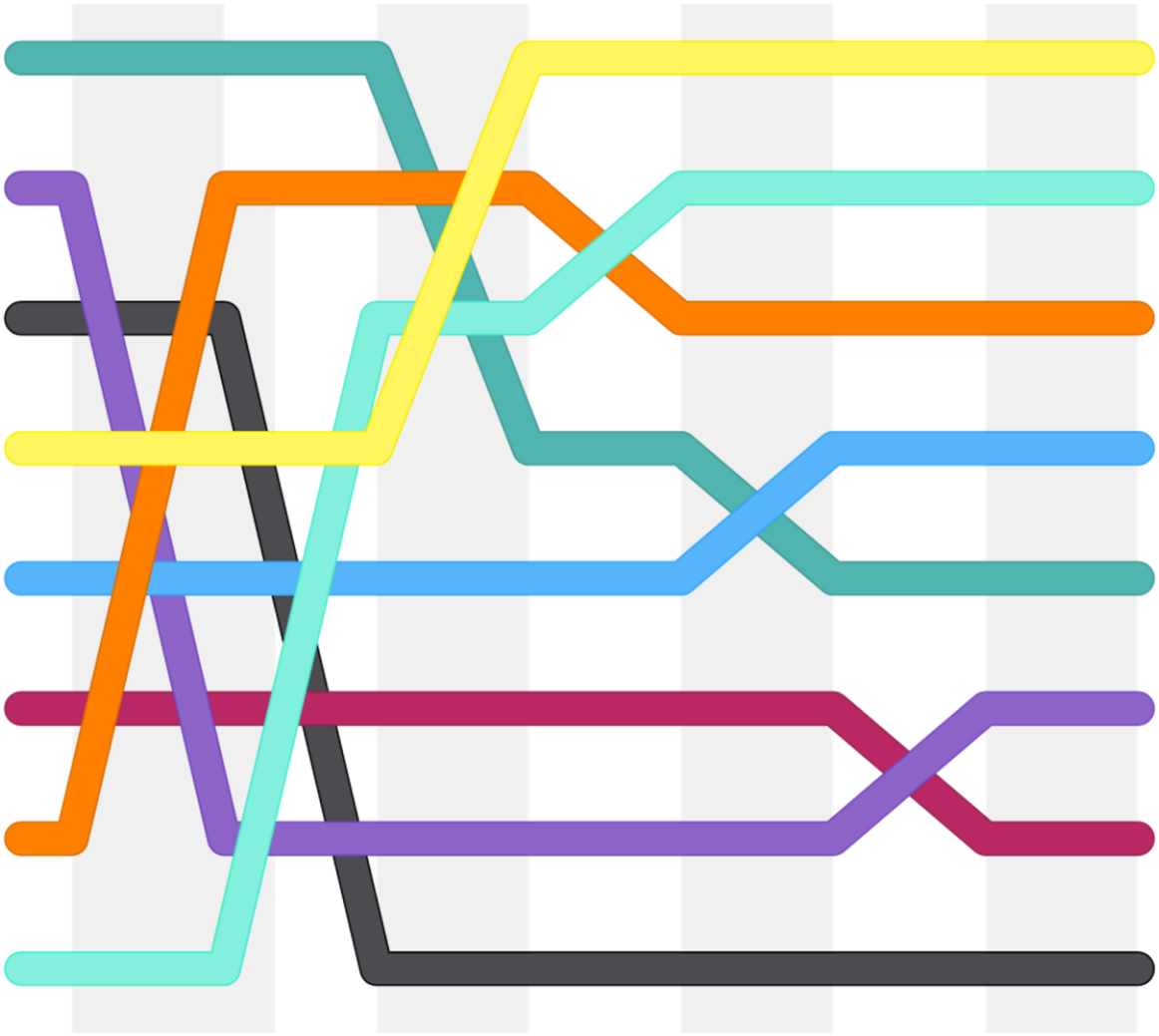


Sorting

CS553| Chetan Gupta |A20378854| 04/11/18

1: Introduction

External data sort and multithreaded programming is performed on single node shared memory for two given datasets of 2 GB and 20 GB each. The purpose of the project is to sort datasets larger than the size of the memory using External Sorting Technique. The results are evaluated and then comparison is carried out with the implementation of the results of Linux sorting. This document illustrates graphically the strategy for implementing the sorting followed by merging. The document also puts forward tabular performance evaluation for the same.

1.1: General Points

A single interface has been created for performing all the test checks based on command line parameters. The sorting of the datasets has been implemented using Java language.

1.2: Testing Platform

The local machine and Neutron cluster has been used for testing codes.

The details have been presented as follows:

2: **Algorithm followed: -**

For sorting the given datasets of 2 GB and 20 GB, the following steps are being taken:

a. The input files are generated.
b. With the number of threads decided, the datasets are divided into smaller chunks of size by inserting into array list.
c. After the assignment into the array lists, a new thread is generated for every sub-chunk.
d. In- place sorting is done.
e. The output is stored in a temp file.
f. Sorting is performed over the temp files.
g. Merging is performed once all the files have been sorted.
h. All the temp files are then deleted.

i. The input data is split into number of chunks in such a way that each chunk fits into main memory. The smaller chunks are then sorted and finally merged together using MERGE SORT algorithm and written back into disk.
j. First, the program takes in three input arguments: file name, number of threads, usable memory in bytes. The generated input file from gensort has lines each of size 100 bytes.

k. Now we will split the large input file into number of small infiles depending on the max_memory, input size and number of threads.

```java
public static void mergeSortedFiles(final List<File> files, final String outputFile, final StringComparator cmp)
    throws IOException
    {

            List<BufferedReader> brReaders = new ArrayList<>();
            TreeMap<String, BufferedReader> map = new TreeMap<>(cmp);

            File f = new File(outputFile);
            if (f.exists()) {
                f.delete();
            }

            BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile, true));

            try
                {
                    for (File file : files) {
                        BufferedReader br = new BufferedReader(new FileReader(file));
                        brReaders.add(br);
                        String line = br.readLine();
                        map.put(line, br);
                    }
                    while (!map.isEmpty())
                    {
                        Map.Entry<String, BufferedReader> nextToGo = map.pollFirstEntry();

                        bw.write(nextToGo.getKey());
                        bw.write("\r\n");

                        String line = nextToGo.getValue().readLine();
                        if (line != null) {
                            map.put(line, nextToGo.getValue());
                        }
                    }
                }
            finally
                {
                    if (brReaders != null)
                    {
                    for (BufferedReader br : brReaders) {
                        br.close();
                    }

                    File dir = files.get(0).getParentFile();
                    for (File file : files) {
                        // file.delete();
                    }
                    // if (dir.exists()) {
                    //     dir.delete();
                    // }
```

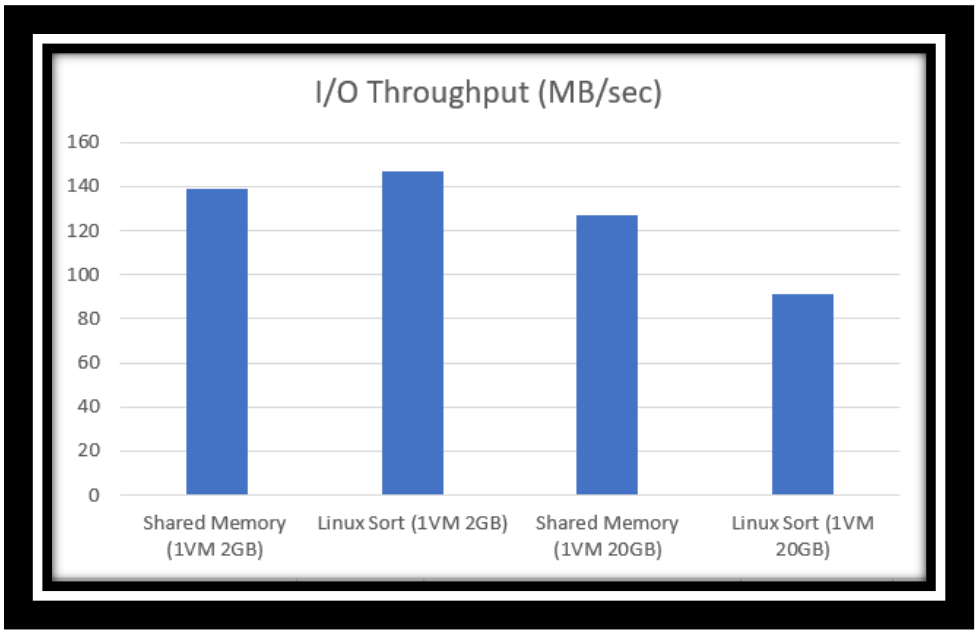
l. Now we will use multi-threading to sort all the infiles and simultaneously removing the unsorted ones.

m. Now we will take chunks of these sorted infiles and sort them with chunks from other infiles and writing to the final output.

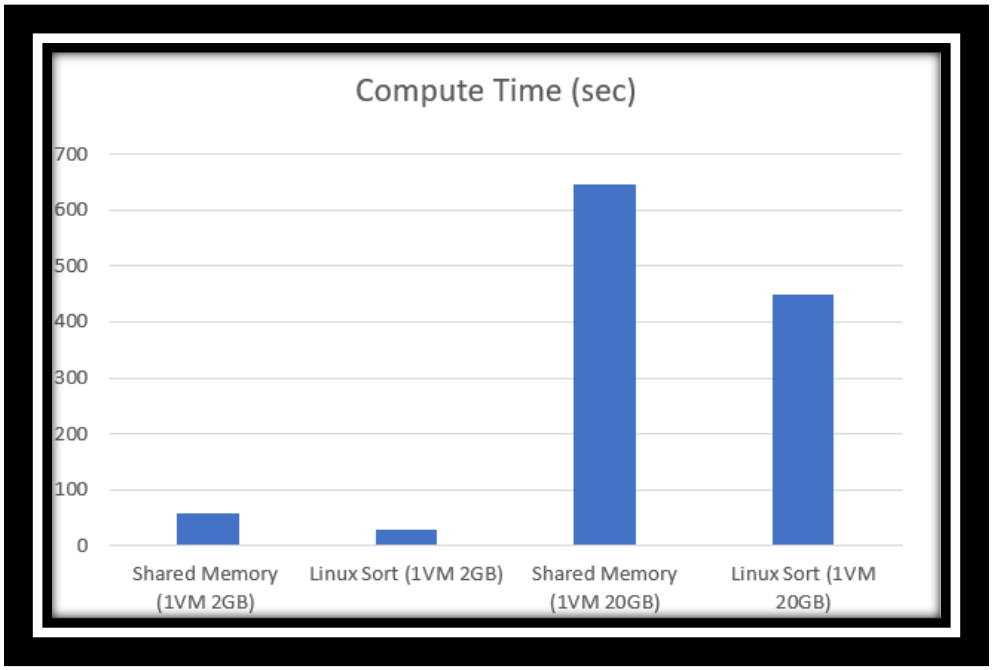
```
public static void main(String[] args) throws Exception
{
    Path fullPath = new File("/input/data-20GB.in").toPath();
    long startTime = System.currentTimeMillis();
    List<File> files = splitAndSortTempFiles(fullPath.toAbsolutePath().toString(), "/tmp/", 25,
            new StringComparator());
    mergeSortedFiles(files, "/tmp/tfinal", new StringComparator());
    long endTime   = System.currentTimeMillis();
        long totalTime = endTime - startTime;
        System.out.println("total time taken is "+totalTime);


}


// Driver program
```

Screen shots for linsort:-

Comparison between Sorting (on Cluster) and Linux Sorting



I/O in MBps



Compute time in Secs

Experiment	Shared Memory (1VM 2GB)	Linux Sort (1VM 2GB)	Shared Memory (1VM 20GB)	Linux Sort (1VM 20GB)
Compute Time (sec)	58.834	27.876	646.486	447.88
Data Read (GB)	4	2	40	20
Data Write (GB)	4	2	40	20
I/O Throughput (MB/sec)	139.2392154	146.9364	126.7158144	91.4530678

Conclusion: - as per data we have received by running the code on cluster we found that Linux sort is faster than the code we have developed.