Comparison and Simulation of External-Sort on a huge dataset

Jayendra Khandare,Graduate Student
School of Informatics and Computing

Hitesh Kumar Dasika, Graduate Student
School of Informatics and Computing

December 6, 2016

1 Introduction

Sorting plays an important task in most of the computer mechanisms. Most importantly, sorting algorithms can be used to solve other problems like counting duplicates, deciding rankings, finding medians in a dataset and many other problems. Commercially talking sorting can be used for tasks like event- driven simulations, searching for some specific information, numerical computations. Various sorting algorithms are in place to handle sorting tasks for a limited size of datasets.

Outline For datasets of huge size, the conventional sorting algorithms could not work since the values to be compared must be on the RAM at all the times. Here, external sorting algorithms come to the rescue.
External sorting algorithms can easily handle large datasets which cannot be sorted by conventional sorting algorithms. Basically, there are many flavors of external sorting algorithms since they are cooked up using a combination of conventional sorting algorithms. The dataset which is too huge to handle is split up in various partitions of sizes which are manageable. These partitions are then sorted using conventional sorting algorithms like merge sort, heap sort, or quick sort and stored at a buffer. Finally, these buffers are

merged again to create the same dataset from which they stem from but in sorted order.

2 Applied algorithm

The algorithm that we have implemented which is also a variation of external sort, divides the dataset into 20 equal parts. This is done for the sake of convenience because it is almost impossible to store a data this big in a specific program variable whether it is a string, an array, or a list. It is fine for smaller databases, but as the size of database increases the sorting algorithm cant store these values in any structure. It may try to do that to a certain level, but once that level is reached the program stops responding.

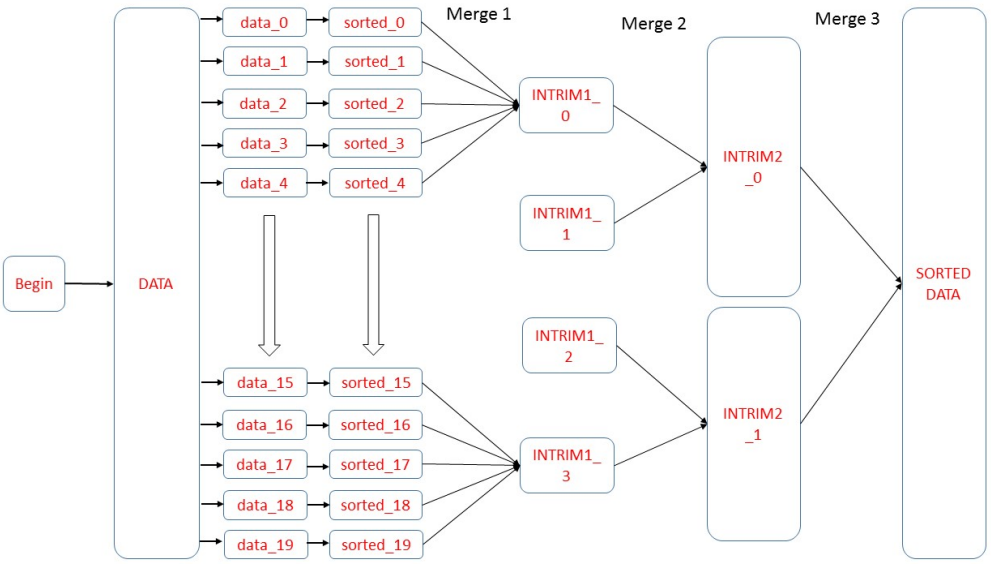

Figure 1: Algorithm Implementation

The 20 partitions created are stored in 20 separate datasets and where they are not so large, hence at this level they can be sent to a sorting program. The dataset so created will be of size $\frac{k}{20}$, considering the original data to be of size k. The sorted data is stored in datasets of equal size. Once the sorting is done, we can combine the data to create the initial dataset but sorted. This phase can take O(log(n)) time.

Although we are making 20 partitions here, the base for the logarithmic function doesnt have to change. The second phase which is sorting can take time depending upon the sorting algorithm that is used. e.g. insertion sort can take O(n^2) time, whereas merge sort can take O(nlog(n)) time which is way faster that insertion sort.

We can do the merge using various methodologies but we have chosen a different approach here. Instead of performing a 20 way merge, we performed a 5 way 4 times, which gives us 4 intermediate datasets. These intermediate datasets which are just buffers used for further calculations. So, this is the first merging scenario which takes place. As the building blocks for this dataset are of size $\frac{k}{20}$ and we are using 5 of the partitions to create an intermediate buffer dataset, its size will be $\frac{k*5}{20} = \frac{k}{4}$.

So, at this stage we have 4 sorted datasets of size$\frac{k}{4}$ each. Once the first level of merging is done, the 4 buffer datasets are merged 2 at a time to provide us with 2 buffers which will be merged for the final output. This two-way merge can be performed in O(n) time and the buffers so created will be of size$\frac{k}{2}$. Basically, these two buffers have half the size of original dataset. The final merge is performed on these two to give the final output as SORTED which can be performed in O(k) time and the final sorted dataset will have size k. That is the size of our original dataset.

3 Implementation

For implementing external sort, I have used Python language, so that the required activities can be split up between modules and maintain the flow of execution. The whole thing can be broken down in six modules with one controller module.

1.MAIN(controller)
2.GENERATE
3.DIVIDE
4.SORT
5.MERGE
6.SIMPLE SORT
7.VARSORTS

In **MAIN** module,all the required modules are called and provided with

necessary variables for functioning. Also this module also calculates the time taken by modules to perform a specific task.
In **GENERATE** module, variable size is provided to the module and the module creates random integers using Python's random.randint() function. The data so created is stored in 'data.csv' file. The number of values created at this phase is equal to variable size provided at the beginning.
In **DIVIDE** module,the dataset created earlier are split in 20 partitions. The partitions so created are 'data0.csv', 'data1.csv', 'data19.csv'. We can change the number of partitions, but the algorithm we are using here requires 20 partitions, hence number of partitions is hard-coded to be 20.
In **SORT** module, the partitions created earlier are stored in an array and that array is sorted using VAR SORTS module. The sorted partitions are stored in csv files as 'sorted0.csv','sorted1.csv','sorted19.csv'.
In **MERGE** module, there are 3 phases. In the first phase, 5 sorted datasets are taken and merged. So, for 20 sorted datasets, we get 4 buffers named 'intrim10.csv','intrim11.csv',etc. In the second phase, two buffer files are taken and merged. So, for 4 buffers, we get two buffers which are named 'intrim20.csv' and 'intrim21.csv'. Finally, these two buffers are merged to created the final output which is stored in 'final sorted data.csv', which has the same values as our initial data but in a sorted manner.
In **SIMPLE SORT** module, the data created from GENERATE module is received and stored in an array. This array is then subjected to Merge Sort algorithm. The output of that is stored in 'simple sorted.csv'.
In **VARSORTS** module, this module contains the coding implementations of some basic sorts like Insertion Sort, Merge Sort and Quick Sort. In this algorithm, we are using Merge Sort as the primary sorting algorithm, but that can be easily changed.

4 Results

NO.	INTEGERS IN DATASET	DATA GENERATION(s)	SIMPLE MERGE SORT(s)	EXTERNAL SORT(s)
1	20	0.00185	0.00134	0.12704
2	100	0.00337	0.00189	0.12571
3	1000	0.01171	0.01011	0.17588
4	10000	0.10326	0.10906	0.44579
5	100000	0.97074	1.26735	3.33871
6	1000000	9.85582	15.09977	36.26858
7	10000000	97.29371	178.16501	382.16395
8	100000000	941.3818	1920.10326	3999.32502
9	1000000000	5256.38994	NONE	73011.35791

We first run the program on the central server provided by School Of Informatics and Computing at Indiana University called Hulk(hulk.soic.indiana.edu). Hulk is a quad-socket, 8-core (32 total cores) AMD Opteron system with 512GB of memory running 64-bit Red Hat Enterprise Linux.The first 8 entries in the table above are Hulk generated. The user is allowed only space upto 10GB at Hulk and size of dataset at entry 8 is 1.25GB.

So, any dataset bigger than 5 GB was not possible to be sorted using Hulk, since the buffers are also to be maintained and they are approximately of the same size as the dataset. Hence for calculations with 1000000000 entries, we used our personal notebook which has 2-cores,1TB secondary memory and 12GB primary memory.Since it is major downgrade, the performance suffered but we were able to make one more result entry i.e. entry 9.

5 Comparative analysis

So basically, it is always preferable to use basic sorting algorithms rather than going for external sort becasue it takes a lot of time comparatively. Also various tasks like reading from a file and storing data in an array take a lot of time which cannot be minimized. It is clear from the results above for data having number of entries less than 100000000, simple merge sort performs very well as compared to external sort on the same data.

But as the number of entries in the dataset increases than 100000000, external sort is able to function properly. But simple merge sort stops responding. This is because for every program operating system allocates some space on the memory. With huge datasets, the number of integers involved are in astronomical terms. Operating system cannot provide a program to have that much memory, so it automatically crashes the program. When same data is given to external sort, it divides it into multiple pieces which can be fit into memory without allowing it to go into overdrive.
In simple terms, if the entries in the dataset are going to be huge, external sorting algorithms should be used for sorting purposes.

6 Platforms used

For this project, we used Python as the main programming language. This project was developed with two different approaches.
In this version of approach, data created was stored in csv files on a Windows machine with standard file system.This approach was handled by Jayendra Khandare. The second approach however used the Postgress database system to store the data and perform all the required activities. This approach was handled by Hitesh Kumar Dasika.

It was found that for smaller dataset, the database approach was faster. But as the size of database increased above 1000000 entries, filesystem approach proved to be faster. Since the query evaluation time in the database system was considerably high with huge databases.

7 Conclusion

This project helped us to understand many important things regarding soting algorithms and why they are needed. We studied various implementations of external sort algorithm and basic sorting algorithms.
But most important thing this project taught us,the fastest is not always the best.

References

[1] Jiamang Wang, Yongjun Wu, Chao Li, and others.
FuxiSort : Paper published as an entry at a competition organised by sortbenchmark.org in 2015. Alibaba Group Inc

[2] External Sorting in geeksforgeeks
`http://www.geeksforgeeks.org/external-sorting/`

[3] Pictorial Representation of External Sorting
`http://www.geeksforgeeks.org/external-sorting/`

[4] Jeffrey Scott Vitter.*Algorithms and Data Structures for External Memory.* Purdue University, West Lafayette.