Multi-threaded Sorting Performance Analysis Using RPCs

Md. Ashekur Rahman
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
mohammad.ashekur.rahman@g.bracu.ac.bd

Asma Ul Hussna
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
asma.ul.hussna@g.bracu.ac.bd

Sanjida Ali Shusmita
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
sanjida.ali.shusmita@g.bracu.ac.bd

Samiha Khan
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
samiha.khan@g.bracu.ac.bd

Iffat Immami Trisha
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
iffat.immami.trisha@g.bracu.ac.bd

Md Sabbir Hossain
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
md.sabbir.hossain1@g.bracu.ac.bd

Annajiat Alim Rasel
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
annajiat@gmail.com

***Abstract*—Distributed procedures that use a message passing approach communicate with one another by transferring bits of information in order to provide unique access to all available tasks.In our paper, we look at sorting the unsorted numbers in distributed systems. Nearly every single modern computer now has a CPU with several cores, which provides additional computing capacity. Parallel execution is critical in the new age of big data to enhance performance to a reasonable threshold. Parallelization brings with it additional issues that must be tackled. Parallel algorithms are studied and contrasted in comparison to it's consecutive equivalents. Determine the opportunity for multi-threading speedup and the elements that influence performance and doing our research for multi-threaded methods with defined time difficulties as well. Following that, an experiment was conducted in which a set of algorithms was implemented and data was gathered through bench-marking and testing. To establish the validity of parallelisation, the data was inspected and analyzed with its associated source code. A review was carried out in order to obtain a better idea of the scope of sorting algorithms and parallel computing. In this research,a multi-sorting threading algorithm approach has been conducted based on previous work. The primary idea behind the algorithm is simple as well as RPCs (remote procedure calls) approach has been used.**

***Index Terms*—sorting, parallelization, algorithms, multithreading, RPCs**

I. Introduction

A distributed system is one in which the components are spread across multiple networked computers and communicate and coordinate their actions by transferring messages from one system to the next.The field of computer science known as distributed computing explores dispersed systems. In order to reach a common purpose, the components interact with one another. In computer science,sorting is among the most basic and quite well discussed topics. A Sorting Algorithm rearranges the items of an array or list based on a comparison operator upon these items. In the respective data structure, the comparison operator is employed to determine the new order of element. When two or more items with the same value keep the same relative positions after sorting, the sorting method is said to be stable [1]. This is also the sorting algorithms which is used by default in the majority of standard libraries [2]. The sorting problem entails arranging N values in sorted order in a distributed system of N processors [3]. Let the values range from 0 to L. On a bidirectional ring with N processors, any sorting method uses O(N 2 lg(L/N)/lg N) messages. On a square mesh with N processors, every sorting method takes [2(N 3n lg(L/N)/lg N) messages. The first lower bound is

achieved using a unique sorting technique for unidirectional rings[3]. A distributed computing system's processors must interact with one another in order to work together to solve a problem.The distributed sorting problem is comparable to a parallel sorting problem on a linear array, which can be addressed using a synchronous model and the odd even transposition sort [4]. Based on prior work, we present a multi sorting threading algorithm method in this paper. The algorithm's main concept is straightforward. We use remote procedure calls (RPCs) on Python. The project's purpose is to develop a sorting algorithm that is both distributed and scalable. The Timsort algorithm has been performed which is a hybrid sorting algorithm that includes merge sort and insertion sort to operate well on a range of real data. This strategy, obviously, has a number of drawbacks. Above all, we are unlikely to be able to construct a flawless estimation procedure. Finally, a comparison has been shown to observe single server performance was enhanced to 33.26%, 34.19% and 34.45%.

II. Literature Review

In this section, we will briefly discuss some existing sorting techniques which were proposed by other authors. Various sorting techniques have been implemented by different authors to achieve better sorting results. Khatami et al. proposed a load-balanced parallel and distributed sorting algorithm [5]. This algorithm implemented with PGX.D. Following algorithm can fetch data efficiently to minimize the overheads in merging and partitioning step by ensuring balanced load. Unique handler used to balance merging for better parallel performance. PGX.D parallel distributed graph analytic framework can handle distributed environments and also ensures balanced workload in machines. PGX.D delay unnecessary computations to reduce communication overhead. Performance of the following algorithm is higher than Spark. Proposed model can handle duplicate data entries more efficiently. Moreover by this sorting technique, high-level API is disclosed to the end user. Prarthana and Karamchandani proposed a model named topk ordering on distributed systems [6]. With the help of a distributed computing system to obtain partially sorted data from a large dataset Coded partial sort method proposed. From a dataset topk ordered instances were found to optimally utilize the servers. Conventional TeraSort algorithm modified to remove those data which are irrelevant for partial ordering. When communication load decreases then the performance improves in Uncoded partial sort. TeraSort is a kind of application of Mapreduce framework. The TeraSort algorithm helps to find the k ordered item list even if complete sorting is not done. For large datasets Coded partial sort can efficiently perform partial ranking. Locally available information observed to estimate the key domain of topk order list. Fallah [7] discussed multi key sorting issues on computer clusters. Distributed and concurrent data structures proposed to perform sort on computer clusters. The author mentioned Alpha Maci Cluster which consists of high speed network connection and interconnected multiple processors. The author performed a multidimensional data sorting algorithm to develop a high performance sorting technique. Alpha trees, alpha-parallel trees, alpha-search trees, alpha-lists and alpha-matrices are distributed and concurrent multikey structures introduced by the author. Single and multi dimensional keys need to maintain lexicographic order. Shiono et al. [8] proposed a method of evaluating the concurrent sorting algorithm. The characteristics of Communicating Sequential Processes (CSP) are optimized to process concurrent embedded systems. Synchronous communication and formal method based verification methods are focused in this study. Bionic sort used to create an efficient sorting network. Inter process communication number is reduced in this model. Zhou et al. [9] mentioned a model to efficiently implement different sorting algorithms on asynchronous distributed memory machines. Merging different elements which are stored in isolated processing elements is quite difficult. The proposed algorithm works efficiently on different distributed machines named as Thinking machines CM5 and Fujitsu AP1000. The efficiency of merging becomes higher with this mechanism. Balanced computational load obtained with this method. Michael [3] discussed the sorting complexity of distributed systems. Author examines the communication cost which is mandatory for arranging N values into sorted order. In a distributed system which consists of N processors, it is difficult to arrange N values into sorted orders. Different sorting algorithms needed some messages on bidirectional rings with N processors. Lower bound for sorting discussed in this model. Shamoto et al. [10] mentioned a GPU based model for large scale sorting problems. HykSort splitter based performed in this study. This following technique can sort multiple elements and in an iterative manner merge them into an array. Sample sort, PSort and Histogram Sort are the forms of Splitter based algorithm. This algorithm decreases the communication cost. The authors perform local sorting, selecting the splitters transferring data between nodes and finally merge those. They iterate transfer-sort-transfer-cycles to sort large data than GPU memory. N. H. Liyanage [11] compares the parallel sorting techniques, architectures and behaviors in a distributed environment. Multiple processor machines along with shared memory focused here. Bionic sort, GPUMemSort, Aligned Access Sort, GPUTeraSort, Parallel Quicksort, Parallel Merge Sort, Parallel Sample Sort discussed in this study. Critical analysis of these sorting techniques performed here. For micro level sorting adopting a sequential or parallel sorting algorithm is required. Karlsson et al. proposed ePUMA architecture for sorting large data sets [12]. For distributed memory architecture fixed size merging task sorting algorithm discussed in this study. For the sake of high efficient local sorting and merging kernels, penalty free unaligned and out of order local memory access is used. In comparison with high performance CPUs and GPUs, this proposed model can achieve high performance. The authors discussed how local multi-bank memories can improve the sorting rate ePUMA based sorting and merging focused here. R. R. Prasath mentioned an alternative time-optimal distributed sorting algorithm on line networks [13]. For the case of distributed sorting,

worst case lower bound along with the efficient algorithm discussed here. Lower bound of (n-1) rounds optimized here. In intermediate processors it does not create copies of (n-2) elements. This model reduced the execution time for Sasaki's time-optimal algorithm and improved the performance. On linear embedding this algorithm can sort distributed elements.

III. Proposed Model

The distributed system in this paper is described as follows: Initially, multiple servers were created using the XML-RPC server module which gives a basic server framework for XML-RPC servers in Python. Each server was assigned their port and host while creation of the server instance. Additionally, remote functions were defined and registered in that server script which can be accessed by clients. In Figure

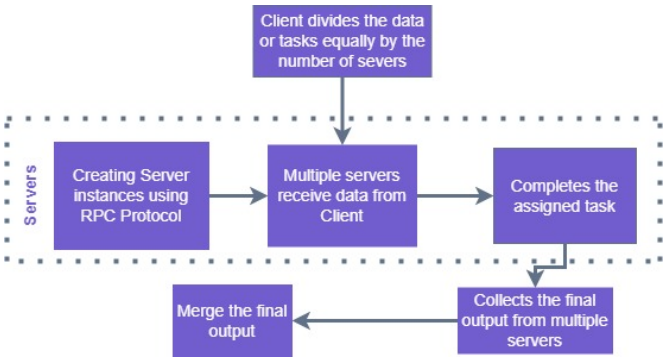

Fig. 1. The work flow of distributed system

1, the data or job from the client-side is divided equally by the number of servers and sent to multiple servers to complete the assigned task. Then after the parallel processing in multi thread, the final output is sent back to the client. This Parallel processing in multiple servers takes less processing time to complete the whole task. Here, We have used sorting

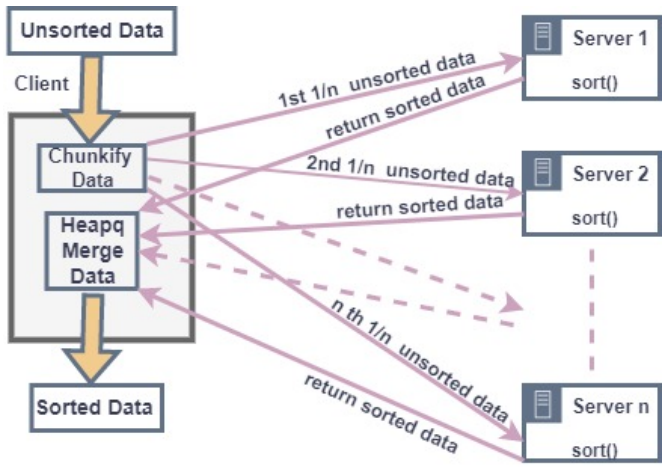

Fig. 2. The process of sorting in distributed system

data as a task to validate our method, if this process works or not? The client-side read a huge dataset of unsorted numbers. After taking the data as an input, it divides the dataset into chunks and sends the divided portions to the active servers to sort and they are returned as sorted lists. Later, those sorted portions are merged on the client-side. This distributed way of sorting reduces the time of sorting drastically when the number of servers increases. The parallel sorting problem in a linear array is quite similar to the sorting problem in a distributed system. According to the model illustrated in figure 2, the unordered list of dataset D is distributed into D_1, D_2, D_3....D_n using chunkify method, where n is the number of active servers in that system. Then these divided portions are sent to the servers. So, the number of data in each part is D_i

$N_i = N/n - (i)$

where N is the number of data in dataset D and

i = 1,2,3,.....n

Here, the servers use a default sort function of Python. This function uses the Timsort algorithm, which is a hybrid sorting algorithm that combines merge sort and insertion sort to perform well on a wide range of real-world data. The time complexity of this Timsort algorithm is,

for single server, $T_1 = O(NLogN) - - - - - - - - - - - (ii)$

for multiple servers, $T_n = O(N/nLogN/n) - - - - - - - (iii)$

According to the above (ii) and (iii)

$T_n < T_1 - (iv)$

From equation (iv), it is clear that this takes less time to sort when multiple servers are used rather than using a single server. After the sorting is completed, the sorted data returns to the client. On the client-side, the sorted data from multiple servers are merged using the heapq merge function. Any number of sorted iterables can be merged using heapq merge. It has an O(NlogK) time complexity, where N is the total number of components and K is the number of items supplied into the minheap for comparison. However, assigning more than required server can cost a lot of time rather than minimizing it. For this, while working with distributed systems, servers need to be assigned according to the dataset or task size.

IV. Dataset Description

The dataset consists of ten million random unsorted numbers where numbers are 1 <= N <= 1000000. As the dataset is comparatively big, It was a perfect dataset to work with and compare the computational time for sorting based on the number of assigned servers.

V. Result Analysis

According to the mentioned process, after taking the dataset as input on the Client-side, we have initialized 1, 2, 3, 4, 5, and 6 servers to compare the performance based on the number of initialized and active servers. In figure 2, when we have sorted the dataset on two servers instead of one server, the processing time decreased drastically from 92.21s to 81.68s.

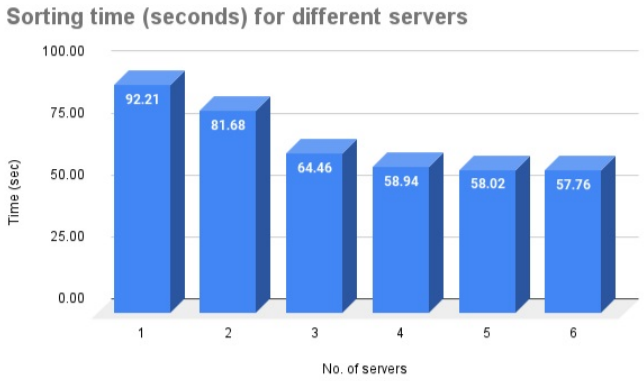

Fig. 3. The sorting time for different server

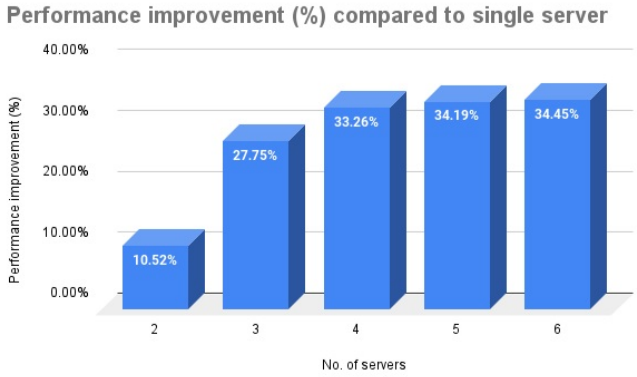

Fig. 4. The single server performance improvement comparison

After adding one more active server(three servers) the sorting time came down to 64.46s and when the client had four servers to sort the unordered dataset, it took less than 58.54s. Similarly, 5 servers and 6 servers were also tested to sort the same dataset but the computational time was decreasing at a slow pace which is 58.02s and 57.76s respectively. 10.52% improvement in performance was noticed from figure 3 when the number of servers was increased from 1 to 2. Similarly, a dramatic improvement of 27.75% in performance was noticed using 3 active servers to sort the random numbers. For 4, 5, and 6 active servers the performance increased to 33.26%, 34.19%, and 34.45% compared to a single server. Similarly, no surprising change was seen when 5 and 6 active servers were there. After analyzing both figures 2 and 3, if the number of active servers increases while sorting a dataset then the computational time decreases which indicates the performance improvement. Another point is that after adding the 4th or 5th number server, little change was noticed. So we should limit the number of active servers based on the size of the dataset to avoid extra data transfer time.

VI. Conclusion

Sorting is the process of rearranging a group of objects into a new order, as well as its objective is to make sorting easier to find further in the set. If information is distributed from input to multiple temporal structures, the sorting process is called a distribution sorting algorithm. In addition, the components are collected and placed in the output using this structure. With a parallel distribution method, distributed sorting is used to manage and process enormous data collections. Furthermore, the distributed sorting problem is similar to a linear array parallel sorting problem, which can be solved with a synchronous model and the odd-even transposition sort. In this research, a multi-sorting threading algorithm approach such as Timsort with RPCs (remote procedure calls) methods have been used. Here, the Remote Procedure Call (RPC), which involves two processes: the calling process (client) and the call-making process (server). These processes are usually located between different objects and may even be located on different virtual or physical machines. Besides, the remote method call is syntactically and semantically like a sequential method call. We have performed this sorting algorithm on six different servers to observe sorting time performance improvements compared to different servers. When the number of servers increases, the distributed sorting method diminishes sorting time.

A. Future work

In the future, we will use the TeraSort algorithm to achieve better sorting results since this helps to find the k ordered item list even if complete sorting is not performed. In addition, to reduce the number of interprocess communications, we would like to use bionic sort, which is used to create an efficient sorting network. To improve the efficiency of our proposed model, we will employ a load-balanced parallel and distributed sorting algorithm (PGX.D), which can handle duplicate data entries and provide high-level API disclosure to the end-user.

References

[1] J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien, "The dynamics of collective sorting robot-like ants and ant-like robots," in *From animals to animats: proceedings of the first international conference on simulation of adaptive behavior*, 1991, pp. 356–365.

[2] A. Kristo, K. Vaidya, U. Çetintemel, S. Misra, and T. Kraska, "The case for a learned sorting algorithm," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1001–1016, ISBN: 9781450367356. [Online]. Available: https://doi.org/10.1145/3318464.3389752.

[3] M. C. Loui, "The complexity of sorting on distributed systems," *Information and Control*, vol. 60, no. 1-3, pp. 70–85, 1984.

[4] R. Prasath, "Algorithms for distributed sorting and prefix computation in static ad hoc mobile networks," vol. 2, Sep. 2010, pp. V2–144. DOI: 10.1109/ICEIE.2010.5559735.

[5] Z. Khatami, S. Hong, J. Lee, *et al.*, "A load-balanced parallel and distributed sorting algorithm implemented with pgx.d," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 1317–1324. DOI: 10.1109/IPDPSW.2017.30.

[6] N. Karamchandani, "Topk ordering on distributed systems," in *Proceedings of the 2018 on Technologies for the Wireless Edge Workshop*, 2018, pp. 21–25.

[7] A. Fellah, "Concurrent and distributed data structures for multikey sorting on computer clusters," in *Proceedings 16th Annual International Symposium on High Performance Computing Systems and Applications*, IEEE, 2002, p. 281.

[8] Y. Shiono, S. Noguchi, M. Mochida, *et al.*, "Efficiency of concurrent processing of sort using csp," in *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, IEEE, 2012, pp. 524–529.

[9] B. B. Zhou, R. P. Brent, and A. Tridgell, "Efficient implementation of sorting algorithms on asynchronous distributed-memory machines," in *Proceedings of 1994 International Conference on Parallel and Distributed Systems*, IEEE, 1994, pp. 102–106.

[10] H. Shamoto, K. Shirahata, A. Drozd, H. Sato, and S. Matsuoka, "Gpu-accelerated large-scale distributed sorting coping with device memory capacity," *IEEE Transactions on Big Data*, vol. 2, no. 1, pp. 57–69, 2016.

[11] N. H. Liyanage, "Comprehensive comparison of parallel sorting techniques, architectures and behaviors which support for distributed environments," in *2017 International Conference on Big Data Analytics and Computational Intelligence (ICBDAC)*, IEEE, 2017, pp. 412–417.

[12] A. Karlsson, J. Sohl, and D. Liu, "Energy-efficient sorting with the distributed memory architecture epuma," in *2015 IEEE Trustcom/BigDataSE/ISPA*, IEEE, vol. 3, 2015, pp. 116–123.

[13] R. R. Prasath, "An alternative time — optimal distributed sorting algorithm on a line network," in *INC2010: 6th International Conference on Networked Computing*, 2010, pp. 1–6.