Enhancing The Efficiency of Radix Sort by using Clustering Mechanism: A Review
Manju Devi¹ Saurabh Charaya²
¹PG Student ²Head of Dept.
1,2OM Institute of Engineering & Technology, Hisar, Haryana, India

Abstract— Sorting is a technique that arranges data in a specific order, whether it be ascending or descending or lexicographical. Various sorting algorithms have been proposed till the time. Radix Sort is one of them. Our aim is to apply clustering mechanism with radix sort. Elements to be sorted are grouped into various clusters on the basis of their distance from each other. For this purpose K-mean clustering algorithm is used. Then these clusters are sorted separately using radix sort. By using this approach efficiency of radix sort has been increased. This Paper is based on the study of various research papers in the field of Radix Sort.

Key words: Unstable Sorting Algorithm, Clustering Mechanism

I. INTRODUCTION
Sorting refers to arranging data within a particular fashion. Sorting algorithms specify way to arrange data within a particular order. Most commonly used orders are numerical or lexicographical order. Today, in the era of computer science, efficient sorting is the key requirement.

Importance of sorting lies within the fact that data searching could be optimized to a very high level if data is stored in the database within a sorted manner. Also after sorting data become more readable. Many sorting algorithms are available which can be used to sort any type of data and of any size.

A. In-place Sorting and Not-in-place Sorting
Sorting algorithms may require some extra space for comparisons and temporary storage of few data elements. The sorting algorithms which do not require any extra space for sorting and sorting is said to be happened in-place, or in other words, within array itself, is called in-place sorting algorithm. Examples of in-place sorting include bubble sort.

But within some sorting algorithms, extra space is required for sorting which may be more than or equal to the space required for elements being sorted. This type of sorting is called not-in-place sorting. Examples of not-in-place sorting include merge sort.

B. Stable and Not Stable Sorting
If a sorting algorithm, after sorting elements, does not change the sequence of similar elements within which they appear, called stable sorting algorithm.

If a sorting algorithm, after sorting contents, changes the sequence of similar elements within which they appear, is called unstable sorting algorithm.

Abstract— Sorting is a technique that arranges data in a specific order, whether it be ascending or descending or lexicographical. Various sorting algorithms have been proposed till the time. Radix Sort is one of them. Our aim is to apply clustering mechanism with radix sort. Elements to be sorted are grouped into various clusters on the basis of their distance from each other. For this purpose K-mean clustering algorithm is used. Then these clusters are sorted separately using radix sort. By using this approach efficiency of radix sort has been increased. This Paper is based on the study of various research papers in the field of Radix Sort.

Key words: Unstable Sorting Algorithm, Clustering Mechanism

I. INTRODUCTION
Sorting refers to arranging data within a particular fashion. Sorting algorithms specify way to arrange data within a particular order. Most commonly used orders are numerical or lexicographical order. Today, in the era of computer science, efficient sorting is the key requirement.

Importance of sorting lies within the fact that data searching could be optimized to a very high level if data is stored in the database within a sorted manner. Also after sorting data become more readable. Many sorting algorithms are available which can be used to sort any type of data and of any size.

A. In-place Sorting and Not-in-place Sorting
Sorting algorithms may require some extra space for comparisons and temporary storage of few data elements. The sorting algorithms which do not require any extra space for sorting and sorting is said to be happened in-place, or in other words, within array itself, is called in-place sorting algorithm. Examples of in-place sorting include bubble sort.

But within some sorting algorithms, extra space is required for sorting which may be more than or equal to the space required for elements being sorted. This type of sorting is called not-in-place sorting. Examples of not-in-place sorting include merge sort.

B. Stable and Not Stable Sorting
If a sorting algorithm, after sorting elements, does not change the sequence of similar elements within which they appear, called stable sorting algorithm.

If a sorting algorithm, after sorting contents, changes the sequence of similar elements within which they appear, is called unstable sorting algorithm.

II. RADIX SORT
Radix Sort is a non-comparison based integer sorting algorithm that sorts data with integer keys by grouping keys by individual digits which share same significant position and value. Because integers could represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers. Database containing integers as well as strings and floats is called heterogeneous database and MRS is a radix sort algorithm developed by Avinash Shukla and Anil Kishore Saxena for sorting heterogeneous data set[6].

Least significant digit (LSD) radix sort and most significant digit (MSD) radix sort are two classifications of radix sort.

A. LSD Radix Sort
Least Significant Digit (LSD) radix sort or Right Radix Sort processes integers starting from least significant digit (or right most digit) and move towards most significant digit.

B. MSD Radix Sort
Most Significant Digit (MSD) Radix Sort or Left Radix Sort processes integers starting from most significant digit (or left most digit) and move towards least significant digit.

C. Efficiency
Radix sort complexity is \(O(wn)\) where \(n\) is the number of keys to be sorted and \(w\) is the word size of the key i.e. number of digits in the key. Sometimes \(w\) may be represented as a constant, which would make radix sort better (for sufficiently large \(n\)) than best comparison-based sorting algorithms, which all perform \(O(n \log n)\) comparisons to sort \(n\) number of keys. However, in general \(w\) cannot be considered a constant because if all \(n\) keys are distinct, then \(w\) has to be at least \(\log n\) for a random-access machine to be able to store them in memory, which gives a
III. CLUSTERING MECHANISM

Our work is to enhance the performance of existing Radix Sort algorithm which could be done in two ways:
- By customizing hardware
- By customizing existing algorithm

Hardware Customization consist of following
1) Addition of Multi-core processors
2) Addition of Primary Memory
3) Addition of Cache

We will do Customization of existing algorithm by using clustering mechanism
1) Create the clusters of Data to be sorted
2) Data at nearest memory location could be clustered (grouped).
3) In this way scheduled processes may be grouped and processed in Batches.
4) By Batch processing the performance would be better as the latency time of CPU would reduce

In our work we will group jobs in clusters for batch processing.

![Fig. 3: Clustering of Data](image)

A. Clustering Process

Suppose we have following lists as jobs in different sector

<table>
<thead>
<tr>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>15</td>
<td>17</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>15</td>
<td>17</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>15</td>
<td>17</td>
<td>28</td>
</tr>
</tbody>
</table>

Suppose K=3
C1=2
C2=12
C3=30

- So cluster according to distance are as follow 12-5>5-2
- So cluster for data point 5 is C1
- So cluster for data point 6 is C1

In same way cluster would be assigned

<table>
<thead>
<tr>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C2</td>
<td>C2</td>
<td>C2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>15</td>
<td>17</td>
</tr>
</tbody>
</table>

Data Member for C1 are 2,5,6
Data Member for C2 are 8,12,15,17

Data Member for C3 are 28,30
So mean of cluster C1 is (2+5+6)/3=4.3
So mean of cluster C2 is (8+12+15+17)/4=13
So mean of cluster C3 is (28+30)/2=29

Now distance would be recalculated with new mean and the cluster of data point would be changed according to new distance

<table>
<thead>
<tr>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>C1</td>
<td>C2</td>
<td>C2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td>17</td>
<td>28</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For example take 8 from C2 cluster
Now recalculate the distance
8-4.3=3.7
13-8=5

So now the distance of 8 from C1 is less than C2 so now 8 would be member of C1

<table>
<thead>
<tr>
<th>Job</th>
<th>Job</th>
<th>Job</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td>17</td>
<td>28</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Again new mean is calculated for all the clusters and elements are rearranged in the clusters based on their distance from new mean. This process continues itself until the elements’ distribution in the clusters is not same as the distribution in the previous step.

In the similar fashion, the data elements to be sorted are grouped into clusters and Radix Sort is applied separately on the clusters. This process will reduce the time taken for sorting and hence efficiency of radix sort will be enhanced.

IV. LITERATURE SURVEY

A. ARL, A Faster In-Place, Cache Friendly Sorting Algorithm

This paper presented ARL which is fast, in-place, unstable, adaptive, recursive left radix (MSD) sorting algorithm. Among these features adaptive and in-place are new features added by its author [Maus]. Its time complexity is O(N*log M) and space requirements are O(N + logM) – where N is the number of integers to be sorted and M is the maximum value sorted. This algorithm dynamically determines number of bits in its digits for each sorting of a segment of the array. It uses permutation cycles within the given array and sorts by internal moves. For optimization of ARL, insertion sort is used for sorting shorter sub-sequences with length less than 20. It is coded in pseudo-Java. The performance of ARL was compared with both with the built in Quicksort algorithm in Java, Arrays.sort(), and with ordinary Radix sorting (sorting from right-to-left). ARL is almost twice as fast as Quicksort if number of integers to be sorted is greater than 100[Maus]. ARL is also faster than Right Radix algorithm when number of integers to be sorted is large (n> 5*10^4). ARL uses half as much memory as Right Radix algorithm. Advantage of ARL is that it performs adaptive and in-place sorting. Disadvantage of ARL is that is does not performs stable sorting [3].

B. A full parallel radix sorting algorithm for multicore processors

This paper presented PARL which is a parallel left radix sorting algorithm which is to be used on ordinary shared memory with multi core machines, that has just one simple statement in its sequential part. It can be seen as a full parallel version of the ARL (Adaptive Left Radix)
sequential sorting algorithm. It uses multithreading in its implementation considering the fact that to start threads in Java costs approximately 2 to 3 milliseconds but it substantially reduces the overhead of sequential execution of small problems. For implementing it, a special design pattern is used for generating a thread pool of k threads (an old idea), the same number of threads as processor cores. PARL is roughly 5-10 times faster than the sequential ARL on a 32(64) core server, 2-4 times faster on a 4(8) core PC, and twice as fast on the two laptops with 2 and 2(4) cores. The sequential ARL is again 3-5 times faster than Arrays.sort (built in Quicksort algorithm in java) on all machines. Thus PARL is 10-30 faster than standard Java Sort for sufficiently large values of n (n is number of integers to be sorted)[5].

C. Review of Radix Sort & Proposed Modified Radix Sort for Heterogeneous Data Set within Distributed Computing Environment

This paper discussed problems of radix sort, brief study of previous works of radix sort & presented new modified pure radix sort algorithm for large heterogeneous data set. This algorithm is implemented on principal of divide & conquers. It was observed that no single method is optimal to all available data sets with varying complexity of size, number of fields, length etc. Thus an attempt was made to select a set of data set & optimize the implementation by modifying the basic algorithm. This algorithm is dependent on the distributed Computing Environment. It was implemented on many core machines. The given heterogeneous list is divided into two main processes one is numeric and other is string. These two processes work simultaneously. Suppose p1, p2 are the two main process. Each process has a unique processor. Process p1 is further distributed in different sub list according to equal length of elements in a list. These lists were sorted simultaneously on the logic of even & odd logic. After sorting, these lists were combined and again main list was sorted. In the case of p2, a pattern was made. Using these unique pattern, the selected strings were got. Among these strings, same strings were provided same numeric values. Now proposed algorithm was applied on these numeric values for sorting the given string. According to the authors, the given algorithm could do much better job over existing sorting algorithms. Both time & space complexities are optimized with their algorithm. Their results had shown an improvement of 10:20% within computational complexity compound with MRS sort & GPU Quick sort[6].

D. Comparison of Bucket Sort and RADIX Sort

In this paper, time usage and memory consumption of Bucket Sort and Radix Sort for different kinds of input sequences has been measured. RADIX sort was of least significant digit version and used counting sort as underlying sorting algorithm. Bucket sort used the insertion sort as its underlying sorting algorithm. The sorting algorithms were compared using six different use cases and three different input sizes. Bucket sort was found faster than radix sort in all cases, but bucket sort uses more memory in most cases. RADIX sort is as quick for unsorted inputs as it is for sorted inputs but bucket sort is quicker for already sorted inputs. Both radix sort and bucket sort were found to be slow for large input ranges[7].

E. Implementing Radix Sort with Linked Buckets using Lsd & Msd and Their Comparative Analysis and Discussion On Applications

In this paper, implementation of radix sort with linked buckets concept has been presented. Linked bucket concept was used to reduce the memory usage for Large Data Set. LSD Radix sort using fixed length buckets, dynamically allocated buckets and linked buckets has been discussed. Also the performance of MSD radix sort in different application areas has been compared.

| Type of \nradix sort | Efficiency in terms of 
and space utilization |
|------------------|--------------------------|
| LSD Radix sort with fixed size array buckets | - The time complexity is N^2  
- The amount of memory space needed for buckets is more.  
- Useful for sorting of integers, characters |
| LSD Radix sort with dynamic arrays buckets | - The time complexity is N^2  
- The amount of memory space needed for each bucket <= total no. of digits.  
- Useful for sorting of integers, characters |
| LSD radix sort with linked buckets | - The time complexity is N^2 but the amount of memory space needed for each bucket is less.  
- Space complexity will be affected by each node for storing address.  
- Useful for sorting of integers, characters |
| Recursive MSD Radix Sort | - The Recursive process affects the time complexity.  
- Highly useful for sorting the strings in a large data set |
| Modified Single Pass MSD Radix Sort with Insertion Sort | - The amount of time required to sort is lesser than O(N^2)  
- Highly useful for a sorting small amount of key values |

Table 1: Efficiency Comparison[8]

V. CONCLUSIONS

After studying the current research work it is analyzed that time complexity and space complexity are the two most important factors of sorting algorithms. Various implementations of Radix Sort have been given till now which reduce the time and space complexity of Radix Sort. The review process is helpful to us in customizing the Radix Sort so that we can enhance its efficiency. It is also helpful in determining the future scope for our work.

REFERENCES

[5] Arne Maus,” A full parallel radix sorting algorithm for multicore processors”, Dept. of Informatics, University of Oslo