Parallelizing Generalized One-Dimensional Bin Packing Problem using MapReduce

Anika
Computer Science & Engineering Department
Thapar University
Patiala, India
thapar.anika@gmail.com

Dr. Deepak Garg
Computer Science & Engineering Department
Thapar University
Patiala, India
dgarg@thapar.edu

Abstract—**Bin packing problem is one amongst the major problems which need attention in this era of distributed computing. In this optimization is attained by packing a set of items in as fewer bins as possible. Its application can vary from placing data on multiple disks to jobs scheduling, packing advertisements in fixed length radio/TV station breaks etc. The efforts have been put to parallelize the bin packing solution with the well-known programming model, MapReduce which is highly supportive for distributed computing over large cluster of computers. Here we have proposed two different algorithms using two different approaches, for parallelizing generalized bin packing problem. The results obtained were tested on the hadoop cluster organization and complexities were estimated thereafter. It is found that working on the problem set in parallel results in significant time efficient solutions for Bin Packing Problem.**

Keywords—MapReduce, Hadoop, Generalized Bin Packing, First Fit Decreasing, Parallelizing.

I. INTRODUCTION

A. Bin Packing Problem

In bin packing, there is a given list of N items of sizes S_1, S_2, S_3,….,S_N such that $0 < S_i <= 1$. These items are needed to be packed in unit sized bins, which are infinitely supplied in an online fashion. The goal is to optimally pack these items in as few bins as possible.

In online version of this problem, the input items are taken as they come, without prior knowledge of the next one. The work is just to take each input item as it come and allocate it the best possible bin, before considering the next input, and move towards an optimal solution. Whereas in offline version, complete information about all the items that are to be adjusted in the unit sized bins is already provided. It is comparatively easier to reach towards an optimal solution in offline version than online, as prior knowledge of all the item sizes is given. Next fit, First fit and Best fit are the three famous online version algorithms used to derive solutions of this problem. First Fit Decreasing and Best Fit Decreasing are the two offline versions.

Next Fit Algorithm: Put the item in either the same or the next new bin, in which earlier item was allocated.

First Fit Algorithm: Scans the bins list from the starting and puts the item in the very first bin in which it can be accommodated else places that item in a new bin.

Best Fit Algorithm: Fit the items in the closest or tightest bins. Here, such combinations are chosen that optimize the allocation procedure and in each bin lowest possible space is left free.

First Fit Decreasing Algorithm: First sorts the given items in non-decreasing order and then allocate bins to these items, in which they first fit, same as in first fit online algorithm.

Best Fit Decreasing Algorithm: First sorts the given items in non-decreasing order and then allocate bins to these items, in which they best fit, same as in best fit online algorithm.

In Figure 1, a list of items and unit sized bins are taken and output results, after applying these algorithms one by one, are shown.

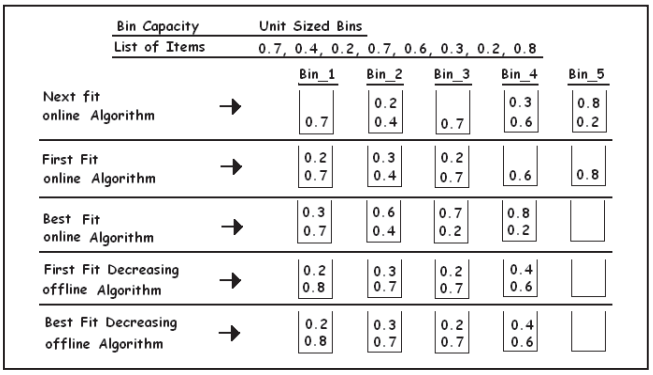

Fig. 1. Example of different Solution of Bin Packing Problem

B. Generalized Bin Packing Problem

Generalized bin packing problem is just same as that of simple bin packing problem, with the only difference that the bins are of variable sizes here. So while allocating the items to the bins, the varying capacities of the bins are also taken into consideration.

Formulating this variable sized bin packing problem:

Here, we are given a list of N items of weights S_1, S_2, S_3,….,S_N such that $0 < S_i < K_1$, $0< i < N$ where K_1 stands for any natural number and act as the item's weight upper bound and a list of M bins of weight W_1,W_2,W_3,….,W_M such that $0 < W_j < K_2$, $0< j < M$ where K_2 stands for any natural number act as bin's weight upper bound. Aim is to pack these items in

these variable sized bins optimally, so that the following condition holds:

$$\sum_i S_i \leq W_j$$

where i = 1,2,…,k.

Considering the real life scenarios, it resembles the situation of a travel agency, which has to pack certain items optimally in its variable sized vehicles. Many other similarities can also be considered in various situations of job scheduling, advertising and many more.

C. MapReduce

Map reduce is a programming model, proposed by Google[1], which helps to parallelize a problem by distributing its large data sets amongst a large number of computers, known as clusters or grids. The main motive of this model is to reduce complexity and help to reduce large dataset problems into smaller ones by dividing it into subparts and thus parallelizing the whole architecture and side by side making the system more efficient and fault tolerant. It also helps to increase the system reliability.

The concept behind the MapReduce is that whole of the computer architecture is divided into 2 types of computers or nodes. One is the master node and rest all are the worker nodes. However worker nodes are further divided into two types according to the tasks assigned to them. Some are the mapper nodes and others are the reducer nodes.

First of all, the master node is initiated/ started. The master node does the work of efficiently distributing the larger problem into smaller sub problems, within the worker nodes. The worker nodes which take in raw data sets are the mapper nodes. They take in these sets and emit out intermediate key value pairs, which consist of data and assigned keys to that data. This phase is also known as the Map phase. The response of intermediate files emitted out by the mapper nodes are further locally shuffled, sorted, grouped and passed on to the reducer nodes . The reducer node collects the intermediate file and helps to bind a particular solution to the given problem, without any redundancy. This phase is also known as the Reduce phase. The output is then collected. Either these act as final outputs or in some cases, these are forwarded further as inputs to the mapper nodes.

Let us take the most discussed example of counting word occurrences to understand its working [1].

Here the task is to count the number of times a word appears in the whole document. The map function takes in the whole document and assigns constant number to each word of the document. Here it is 1 as each word, independently, has been seen only once. The reduce function takes in these intermediate values and increments the count of the respective word based on the number of occurrences of that word. So the final output provided is the total count of each distinct word present in the input document. Figure 2, illustrates this concept with the help of an example.

II. Literature Review

Bin packing problem is a combinatorial NP-hard problem. Various algorithms have been worked out for obtaining an optimal solution by limiting its upper bounds [5][6]. The offline versions of algorithms are best ones and provide an optimal solution in comparatively less time. The online versions are more practical but are far away from the optimal solution. So a number of algorithms have been proposed to achieve optimality in less time. Epstein et al have discussed and compared the online algorithms of the Bin Packing Problem [7][8]. Various improvements in the online version has also been discussed. Gyorgy et al propose one by introducing On-line Sequential Bin Packing [9]. However further improvements in the form of approximation algorithms are also there [10][11].

A major breakthrough in the era of distributed computing i.e. the MapReduce was first of all proposed by the Google. It was further inspired by the map and reduce primitives present in Lisp and many other functional languages [1]. Open Source Software Framework Hadoop is also available that enables applications to work with thousands of computation-independent computers and petabytes of data. Hadoop was derived from Google's MapReduce and Google File System (GFS) papers [2][3].

Since the first paper of MapReduce was launched[1], major advancements in the above field have taken place. It has been used in a wide variety of applications. In Text Processing it can be used in applications like Batch Text Similarity Search with MapReduce, Entitytagger etc. [12][13]. In case of Machine Learning, it has applications like COMET [14], Parallelized K-Means clustering algorithm [15].

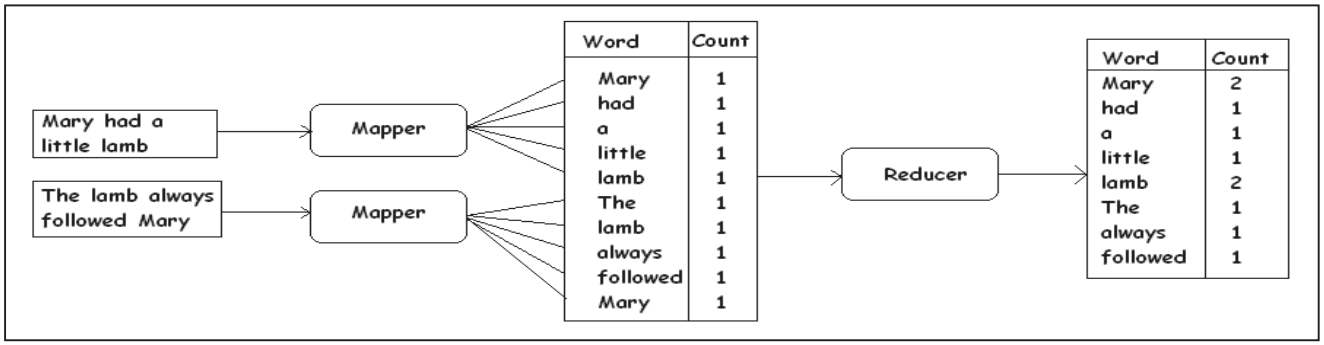

Fig. 2. Sample example of Word Count Using MapReduce

In case of graphs, filtering approach can be used to solve large dense graph problems with smaller and manageable input sizes in the distributed environment using MapReduce [4]. DOULION [16] and HADI [17] are some more applications of the graphs. As it is one of the major providers of distributed computing environment in this era of cloud computing, many more research areas of its usage are continuously evolving.

III. Proposed Algorithm

A. Generalized Bin Packing Using First Fit Decreasing

Here, efforts have also been made to extract the benefits of offline version and applying them for the online version. Here it is assumed to have a window or a buffer of size x. All the input items are one by one fed into this window and when the window is full, the required items are submitted as jobs to the master node of the MapReduce Framework. By the time the required tasks are carried over on MapReduce Framework, the window starts collecting the input items and thus next job inputs are again made ready to be processed. Thus it helps to give an offline feel while using the online version.

The proposed algorithm undergoes the following basic stages:

1. Collection of Inputs: The buffer or window collects the required items along with the bin capacity and count in which these weights are to be adjusted and when the window size is full, the inputs are submitted as first parameter being the bin capacity, then number of bins with the same capacity that are available i.e. bin's count and at last the list of items containing weights that are to be adjusted in those bins, to the Master node of the MapReduce Framework.

2. Mapping of Items: As the inputs are fed into the mapper nodes, each mapper node starts its processing independently. The list of items submitted for a particular sized bin capacity are extracted out and sorted in non-decreasing order. The output of this phase is the bin capacity as the key, then bin's count and finally the non-decreasing ordered list of items that are to be accommodated in those bins.

3. Combining The Mapped items lists: Now the above mapper phase outputs are locally combined, shuffled and sorted according to their key values. And again they are partitioned further depending upon the number of reducers. These lists act as inputs for the reducer nodes.

4. Optimal allocation phase: The reducer nodes are given these mapped items list as inputs. Reducer nodes are given the task of applying first fit decreasing algorithm. As the input items for a particular bin are already sorted, this gives an ease to the reducer. The reducer nodes directly act on these sorted files applies first fit algorithm in order to obtain nearly optimal solutions. The output of this phase is the optimal list of items accommodated in the corresponding bin against which those were submitted.

5. Analyzing the Outputs: The output list now finally contains the following information. First is the bin capacity and then the list of optimally allocated items to that bin. Figure 3 represents the proposed algorithm and Figure 4 shows the sample example.

```
Map Phase
//Input: Bin-Capacity, Count, Items List
For each Bin-Capacity sort the Items list in
Non-decreasing Order using Quick sort
//Output: Bin-Capacity, Count, Non-decreasing
          order sorted Items List

Reduce Phase
//Input: Bin-Capacity, Count, Non-Decreasing Order
         sorted list of Items
1. create an empty Bin's item list
2. While (Count!= 0)
3.      Initialize the Bin's item list to 0
4.      Search for the very first combination of items
        that can be accomodated in the bin without
        exceeding its capacity
5.      append the items to the Bin's item list
6.      remove the items from the original Item list
7.      Emit the (Bin-capacity, Bin's Item List)
//Output: Bin-Capacity, List of Optimally allocated
          items for the respective Bin
```

Fig. 3. Proposed Algorithm for Generalized Bin Packing using First Fit Decreasing

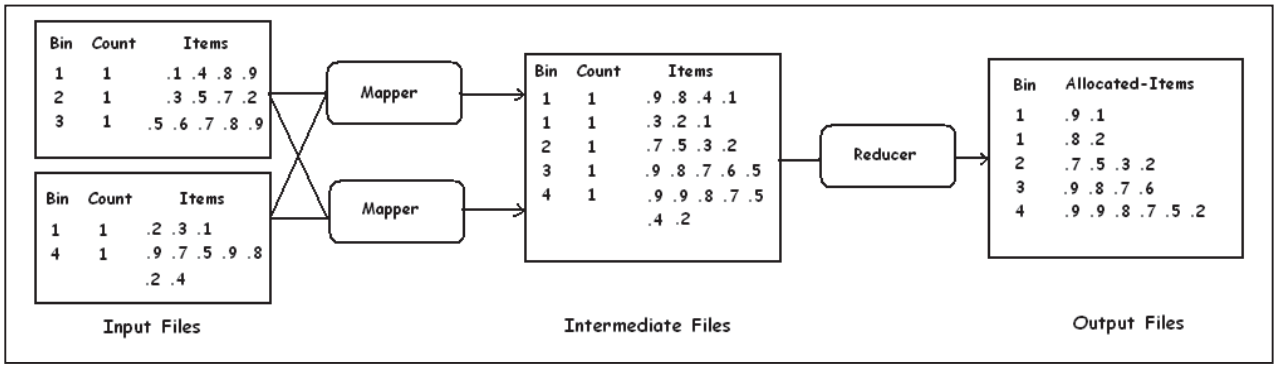

Fig. 4. Sample Example of Generalized Bin Packing using First Fit Decreasing Approach

B. Generalized Bin Packing Using Dynamic Programming

The only difference from the first algorithm is that instead of first fit decreasing, dynamic programming is used here. So, when buffer or window is filled up to an optimal level of bins and their respective items lists, these files are fed as inputs to the Master node for further processing and the window is again allowed to resume their collection work again. However, here instead of First Fit Decreasing algorithm, Dynamic programming approach is implemented in the reducer node.

The proposed algorithm undergoes the following basic stages:

1. Collection of Inputs: The window or buffer or input files collect the input in the following format. Firstly, there is the bin capacity, then the total number of bins of this capacity i.e. its count and finally the list of items that need to be accommodated in the respective bin. After an appropriate collection of inputs, these files are fed as input files to the master nodes.

2. Mapping of Items: As the master nodes take over the control, first task is to divide the task amongst the total number of mappers present. After that each mapper does the task assigned to it independently. In each Map Phase, the total list of items accompanying the bin, is scanned and sorted in ascending order. The output of this phase results in the list of bins, along with their counts and sorted items.

3. Combining The Mapped items lists: Now the output lists of mappers are locally combined, shuffled and sorted according to their key values. These lists are further partitioned depending upon the number of reducers.

4. Optimal allocation phase: The output lists of the mapper phase, after shuffling, sorting of the key elements are fed as input to the reducer phase. Here also, each reducer performs its task independently. For each key element, each reducer first collects the sorted list of items, and then finds the best optimal solutions (equal to number of counts) using dynamic programming approach, and finally emits each optimal solution as output.

5. Analyzing the Outputs: The output list now contains firstly, the bin capacity and then the list of optimally allocated items to that bin. In Figure 6, sample example for this Generalized Bin packing problem is shown and Figure 5 shows the respective algorithm.

```
Map Phase
//Input: Bin-Capacity, Count, Items List
For each Bin-Capacity sort the Items list in
ascending Order using Quick sort
//Output: Bin-Capacity, Count, ascending
          order sorted Items List

Reduce Phase
//Input: Bin-Capacity, Count, ascending Order
         sorted list of Items
1. create an empty Bin's item list
2. While (Count!=0)
3.      Initialize the Bin's item list to 0
4.      Search for the very first combination of items
        that can be accomodated in the bin without
        exceeding its capacity using dynamic programming
5.      append the items to the Bin's item list
6.      remove the items from the original Item list
7.      Emit the (Bin-capacity, Bin's Item List)
//Output: Bin-Capacity, List of Optimally allocated
          items for the respective Bin
```

Fig. 5. Proposed Algorithm for Generalized Bin Packing using Dynamic Programming

IV. RESULTS

These algorithms were tested by running them over the hadoop cluster. The hadoop cluster comprised of at most 10 machines. For these series of experiments the systems that were used had the following configuration Intel(R) Core (TM) i3 CPU M 380 @ 2.53GHz, RAM: 4GB, HDD: 320GB and Operating System: Linux. The algorithms were run on these machines one at a time, side by side increasing the number of nodes in the cluster starting from 2 to 5 & 10 as well as the experimental data set starting from 10 Kb to 100 Mb. There are 10 machines in total, where 1 acts as the master and others as slaves. Firstly, the experiments were conducted by keeping the settings as default i.e. 2 mapper-reducer slots per node. Afterwards, settings were customized to 5 mapper-2 reducer slots per node and 2 mapper-5 reducer slots per node. The input file consisted of bins capacity ranging from 1 to 100 and items also ranging from 1 to 100.

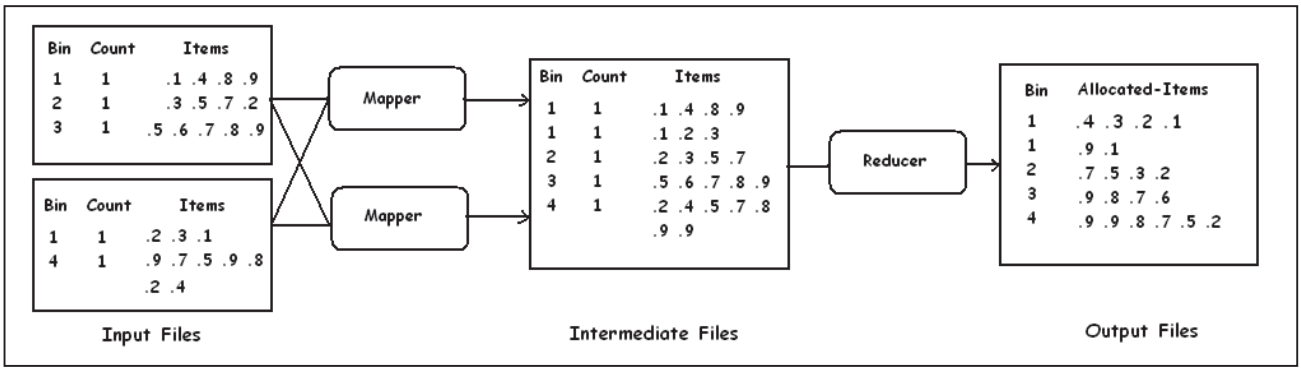

Fig. 6. Sample Example of Generalized Bin Packing using Dynamic Programming Approach

However organization of each file of input file consisted of 1 bin with its capacity and the list of items that need to be accommodated in that bin. The execution times of the algorithms were tracked and approximate results of worst case complexities were obtained. Table I shows the sequential running time for both the proposed algorithms on a single machine.

TABLE I. SEQUENTIAL RUNNING TIME OF BOTH THE ALGORITHMS

Size (in KB)	First Fit Decreasing (in sec)	Dynamic Bin Packing (in sec)
10	.135	450.37
50	26.65	2153.2
100	83.45	-
500	1378.79	-
1000	-	-

Table II shows the running timings of First proposed algorithm on a hadoop cluster with 2 Mapper – 2 Reducer slots per machine.

TABLE II. RUNNING TIME OF FIRST FIT DECREASING ALGORITHM WITH 2 MAPPERS-2 REDUCERS /NODE, ON HADOOP

Size (in KB)	No. of Nodes	First Fit Decreasing (in sec)
10	1	9
	5	10
	10	12
50	1	20
	5	13
	10	17
100	1	22
	5	20
	10	17
500	1	31
	5	25
	10	22
1000	5	56
	10	40
5000	5	173
	10	154
10000	5	528
	10	418
100000	5	1034
	10	889

Table III shows the running timings of Second proposed algorithm on a hadoop cluster with 2 Mapper – 2 Reducer slots per machine.

TABLE III. RUNNING TIME OF DYNAMIC BIN PACKING ALGORITHM WITH 2 MAPPERS-2 REDUCER/NODE, ON HADOOP

Size (in KB)	No. of Nodes	Dynamic Bin Packing (in sec)
10	1	11
	5	13
	10	16
50	1	25
	5	18
	10	27
100	1	23
	5	17
	10	15
500	1	96
	5	85
	10	75
1000	5	314
	10	289
10000	5	679
	10	614
100000	5	1165
	10	960

Table IV shows the running timings of both the algorithms on a hadoop cluster with 5 Mapper – 2 Reducer slots per machine keeping the node size stable.

TABLE IV. RUNNING TIME OF DYNAMIC BIN PACKING ALGORITHM WITH 5 MAPPERS-2 REDUCER/NODE, ON HADOOP

Size (in KB)	First Fit Decreasing (in sec)	Dynamic Bin Packing (in sec)
10	40	53
50	34	46
100	15	21
500	26	84
1000	49	112
10000	189	190
100000	461	510

Table V shows the running timings of both the algorithm on a hadoop cluster with 2 Mapper – 5 Reducer slots per machine keeping the node size stable.

TABLE V. RUNNING TIME OF BOTH THE ALGORITHMS WITH 2 MAPPERS-5 REDUCER/NODE, ON HADOOP

Size (in KB)	First Fit Bin Packing (in sec)	Dynamic Bin Packing (in sec)
10	11	11
50	12	18
100	18	15
500	103	75
1000	359	289
10000	671	606

The results obtained were plotted with x-axis as input file (in Kb) and y-axis as time taken (in seconds). Figure 7 represents the sequential running time of both the algorithms. Figure 8, 9, 10, 11 represents the graphical analysis of these algorithms on the basis of observations, while running in parallel on hadoop.

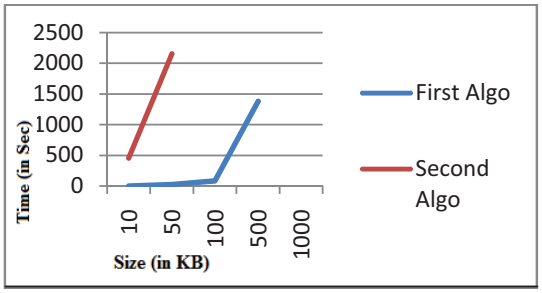

Fig. 7. Graphical representation of Sequential running time of both the algorithms corresponding to Table I.

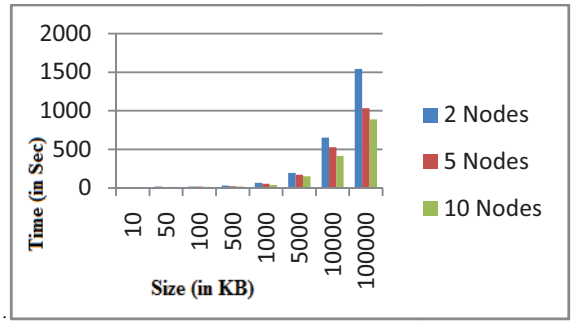

Fig. 8. Graphical representation of First Fit decreasing Bin Packing algorithm corresponding to Table II.

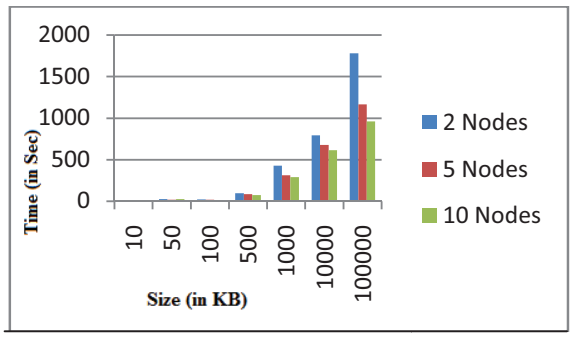

Fig. 9. Graphical representation of Dynamic Bin Packing algorithm corresponding to Table III.

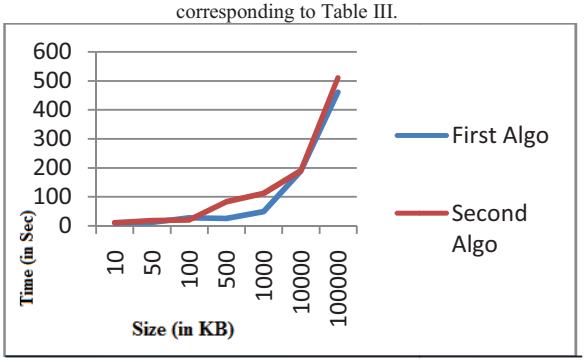

Fig. 10. Behavior of both the algorithms on increasing the number of mappers by keeping the nodes number stable.

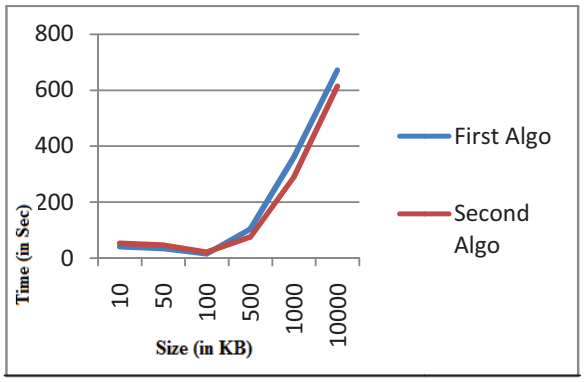

Fig. 11. Behavior of both the algorithms on increasing the number of reducers by keeping the nodes number stable.

V. Discussion

Analyzing the complexity of first proposed algorithm which uses First Fit Decreasing concept, it is found that the Map phase, as it sorts the complete list of input items for each key or bin through quick sort (Figure 3), so its upper bound complexity is estimated to be O (n log n) where n are the number of items per key element. In the Reducer phase, as work is only to allocate those non-decreasing list of items to a specific key or bin, using First Fit decreasing, The overall complexity of reducer phase as per key element is O (n). Now suppose there are total d bins or key elements. So, the complete sequential complexity of both the cases is estimated to be:

d (O(n log n)+O(n))

or O(d n log n)+O(d n)

or considering only the dominant part, we have the complexity as:

O (d n log n).

Now, in parallel programming, considering the case of m-Mappers & r-Reducers and N nodes on hadoop framework, the complexities are found to be as follows:

As the complete work is divided into m-Mappers on N nodes, the complexity varies as: d (O (n log n/N m))

Similarly in case of r-Reducers, the complexity for the reducer phase is as: d (O (n/ N r))

Combining both the parallel complexities, the total complexity is estimated as follows:

O ((d n log n)/N m)+ O ((d n)/N r)

O ((d n log n r)+(d n m)/(N m r))

(d n log n r) being a dominating factor, so rounding it off,

O ((d n log n r)/(m r N))

O ((d n log n)/Nm)

O (c1* n log n)

where constant c1=(d/Nm).

Now, it can be said that running time is inversely proportional to the number of mappers on N nodes. Here by varying the number of mappers on N nodes along with the load on each mapper, running time varying almost linearly with the input size can be obtained.

Now, analyzing the complexity of Second algorithm which uses dynamic programming, it is found that the Map phase, as it sorts the complete list of input items for each key or bin through quick sort(Figure 5), so its upper bound complexity is estimated to be O (n log n) where n are the number of items per key element. In the Reducer phase, as work is only to allocate those non-decreasing list of items to a specific key or bin, using dynamic programming, The overall complexity of reducer phase as per key element is O (n^2). Now suppose there are total d bins or key elements. So, the complete sequential complexity of both the cases is estimated to be:

d (O(n log n)+O(n n))

or O(d n log n)+O(d n n).

Now taking the case parallel programming, for m-Mappers & r-Reducers, we have:

As the complete work is divided into m-Mappers on N nodes, the complexity varies as: d (O (n log n/N m))

Similarly in case of r-Reducers, the complexity for the reducer phase is as: d (O (n n/N r))
Combining both the parallel complexities, the total complexity is estimated as follows:
O ((d n log n)/N m)+ O ((d n n)/N r)
O ((d n log n r) + (d n n m)/(N m r))
O(d n n m) being a dominating factor, so rounding it off,
O ((d n n m)/(N m r))
O ((d n n)/N r)
O (c2 n n)
where constant c2=(d/Nr).
Here, by increasing the number of reducers on N nodes along with the load on each reducer, good results can be obtained in comparatively less time.

However, when the complexities of Map and Reduce phase of each proposed algorithm were compared, the graphs were obtained as in Figures 8, 9, 10 & 11. Some important facts were drawn from here, which are listed in the Table VI. The running time gathered from the hadoop jobs, is far better as compared to their sequential counterparts. Increasing the number of nodes had a significant effect on both the algorithms. So, the factor (d/N) plays an important role. On increasing the number of mapper slots per node, the time complexity was further reduced at a significant rate as factor (d/Nm) came into the play. On increasing the number of reducers, Table V shows that Second algorithm took less time than the first. As discussed earlier, this is an expected behavior of the algorithm.

For smaller input files some violation in complexities can be easily seen. The reason behind it is clear, as there is not enough data to be consumed by all the nodes and as a result the extra time consumed is the overhead cost of the idle machines. Secondly, while comparing Table IV and V it can be seen that running time on increasing the number of reducers is greater than while increasing the number of mappers. This fact reveals the dependence of results on d i.e. the total number of bins. There is one relation that holds from the above observations. The total number of reducers launched must be less than or equal to the number of key elements submitted for the jobs i.e. d >= r. The optimal case exists when d = r. However, when d < r, the overhead cost is increased.

TABLE VI. TABULAR REPRESENTATION OF THE COMPARISON OF BOTH THE ALGORITHMS

	First Proposed Algorithm	**Second Proposed Algorithm**
Approach used	First Fit Decreasing	Dynamic programming
Sequential Time Complexity	O(d n log n)+O(d n)	O(d n log n)+O(d n n)
Parallel Time Complexity	O ((d n log n r)+(d n m)/(N m r))	O ((d n log n r) +(d n n m)/(N m r))
Type of Solution	nearly-optimal	Optimal
Nature	Mapper Specific Computation	Reducer Specific Computation
Complexity determining factor (d=total number of keys)	c1=(d / N m)	c2=(d / N r)
Effect of increasing the number of nodes with 2 Mapper-2 Reducer slots per machine	Time complexity was reduced at higher pace.	Time complexity was reduced at higher pace.
Effect of increasing the number of mappers from 2 to 5 Mapper-2 Reducer slots per machine (N is constant)	Time complexity reduced at a higher pace.	Time complexity reduced at a higher pace.
Effect of increasing the number of reducers with 5 Mapper-2 Reducer slots per machine (N is constant)	Not much effect (instead after a certain number, Time Complexity increased)	Responded well and had good effects on time complexity, but up till a certain number only.
Effect of increasing the number of keys (all other factors being constant)	Time Complexity was reduced (provided no. of mappers and reducers on N nodes are in such an ample proportion to handle the data).	Time Complexity was reduced (provided no. of mappers and reducers on N nodes are in such an ample proportion to handle the data).

VI. CONCLUSION

After the complete analysis of the proposed algorithms, some important conclusions can be drawn. The above algorithms were found to be efficiently working in MapReduce framework. The number of mappers and reducers on each node, need to be adjusted accordingly. Larger the number of nodes, mappers and reducers, larger is the work division and more time optimal solution can be obtained. As here it can be seen that, time complexities obtained here are comparable to the linear time complexities.

However, the main limiting factor in case of the above algorithms is the choice of number of reducers. As the number of reducers cannot be increased beyond a certain limit, the algorithm may also result in the bottleneck problem. That is why a proper ratio of mapper-reducer slots per node and number of key elements need to be maintained, and if it is maintained, the time efficiency will

automatically be incorporated in case of large input files. Future work in this area will be completely focused on incorporating more time efficiency in the solutions of Generalized Bin Packing Problem.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters", Communications of The ACM – 50th anniversary issue: 1958-2008, Volume 51 Issue 1, January 2008, pp. 107-113.

[2] "Welcome to Apache™ Hadoop®!", Welcome to Apache™ Hadoop®!", [Online]. Available: http://www.hadoop.apache.org (Accessed: 2013, November 3).

[3] "Apache Hadoop", FrontPage: Hadoop Wiki, [Online]. Available: http://www.wiki.apache.org/hadoop (Accessed: 2013, November 4).

[4] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri et al, "Filtering: A Method for Solving Graph Problems in MapReduce",SPAA'11 Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures, San Jose, California, USA, June 4–6, 2011, pp. 85-94.

[5] David S. Johnson, Alan J. Demers, Jeffrey D. Ullman et al, "Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms", SICOMP, 1974;Volume 3, Issue 4,pp. 299–325.

[6] David S. Johnson and Michael R. Garey. "A 71/60 Theorem for Bin Packing", Journal of Complexity, 1985; 1,pp. 65-106.

[7] Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt, "Comparing Online Algorithms for Bin Packing Problems", Journal of Scheduling, February 2012, Volume 15, Issue 1, pp. 13-21.

[8] Leah Epstein, Rob van Stee. "Optimal Online Algorithms for Multidimensional Packing Problem". SIAM Journal on Computing, Volume 35, Issue 2, pp. 431–448.

[9] AndrasGyorgy , Gabor Lugosi , Gyorgy Ottucsak, "On-Line Sequential Bin Packing", Journal of Machine Learning Research, 3/1/2010; Volume 11, pp. 89-109.

[10] Richard J. Anderson, Ernst W. Mayr, Manfred K. Warmuth. "Parallel Approximation Algorithms for Bin Packing", Information and Computation, September 1989; Vol. 82, No. 3, pp. 262-277.

[11] Coffman, E. G., Jr. and G. S. Lueker, "Approximation Algorithms for Extensible Bin Packing", SODA '01 Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, pp. 586-588.

[12] Rui Li, Li Ju, ZhuoPeng, Zhiwei Yu, Chaokun , "Batch Text Similarity Search with MapReduce", Web Technologies and Applications, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011; Volume 6612, pp. 412-423.

[13] Kaushik Chakrabarti, Surajit Chaudhuri, Tao Cheng et al, "EntityTagger: Automatically Tagging Entities With Descriptive Phrases", WWW 2011, Proceedings of the 20th international conference companion on World wide web, pp. 19-20.

[14] Justin D. Basilico, M. Arthur Munson, Tamara G. Kolda et al, "COMET: A Recipe for Learning and Using Large Ensembles on Massive Data", Proceedings of the 2011 IEEE International Conference on Data Mining(ICDM), 2011; pp. 41-50.

[15] Likewin Thomas, KiranManjappa, Annappa B et al, "Parallelized K-Means Clustering Algorithm for Self AwareMobile Ad-Hoc Networks", ICCCS'11, Proceedings of the 2011 International Conference on Communication, Computing & Security, pp. 152-155.

[16] Charalampos E. Tsourakakis, U Kang et al, "DOULION: Counting Triangles in Massive Graphs with a Coin", Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 837-846.

[17] U. Kang and Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, Jure Leskovec, "HADI: Mining Radii of Large Graphs", ACM Transactions on Knowledge Discovery from Data, February 2011; Vol. 5, No. 2, Article 8.