International Journal of Engineering & Technology, 7 (4.10) (2018) 157-162

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

An Effective Compact Representation Model for Big Sparse Matrix Multiplication

G. Somasekhar[1], K. Karthikeyan[2,*]

[1]*School of Computer Science and Engineering, VIT, Vellore-632014, Tamil Nadu, India.*
[2]*Department of Mathematics, School of Advanced Sciences, VIT, Vellore-632014, Tamil Nadu, India.*
**Corresponding author E-mail: k.karthikeyan@vit.ac.in*

Abstract

The importance of big data is increasing day by day motivating the researchers towards new inventions. Often, a small amount of data is needed to achieve a solution or to draw a conclusion. The new big data techniques stem from the necessity to retrieve, store and process the required data out of huge data volumes. The present paper focuses on dealing with sparse matrices which is fre-quently needed in many sparse big data applications nowadays. It applies compact representation techniques of sparse data and moulds the required data in the mapreducible format. Then the mapreduce strategy is used to get the results quickly which saves execution time and improves scalability. Finally we established that the new algorithm performs well in sparse big data scenario compared to the existing techniques in big data processing.

Keywords: *Big data; Hadoop; Mapreduce; Matrix multiplication; Sparse data; Sparse matrix.*

1. Introduction

This Nowadays, due to the evolution of social networking websites, mobile devices, sensors and cloud computing, the data is rapidly being incremented from time to time. This enormous data for which traditional RDBMS techniques become inapplicable is termed as big data. Big data processing and big data analysis are the challenges faced by the society today. New techniques, algorithms, and solutions in big data domain are the most desirable now. In the recent past, sparse big data attracted the research community to innovate several big data techniques for its retrieval, storage and efficient processing. Column data compact representations were given by D.J.Abadi [1]. The compact representation techniques for sparse matrices in GPUs were given by B. Neelima and S.R.Prakash [2]. Though sparse matrix multiplication technique proposed by R. Yuster and U. Zwick [3] is fast executing, it is theoretical only. Many techniques in big data were publicized by J. Dean, S. Ghemawat and T. White [4-5]. The Parallelisation and indexing were applied by A. Buluc and J.R. Gilbert [6] in multiplying sparse matrices. The problems due to communication overhead in multiplying sparse matrices were partially solved by G. Ballard et al. [7]. The Parallelisation was implemented by S. Smith et al. [8], in sparse tensor matrix multiplication. The approaches [6-8]cannot be applied to big data. The programmer should be very cautious in distributing the data, balancing the load among nodes in the cluster, replicating the data blocks, communicating information between nodes and so on. Some massive matrix multiplication techniques based on mapreduce were innovated [9-12]. Rather than the advantage of good scalability, the iterative approaches based on HAMA[9-10] waste much of the execution time by taking multiple rounds in multiplying matrices. In the article [11], The chain multiplication problem of matrices was solved by J. Myung and S. Lee, who used the relation to represent a matrix. But they ignored the redundancy problem. Maintenance of intermediate data between rounds is more difficult in the multi-round matrix multiplication [12]. The Vector Linear Combination Approach (VLCA) is stated by Zheng et al. [13]. But it does not use any special sparse matrix input format or layout. In VLCA, null values are not removed from sparse matrices. A considerable number of unnecessary multiplication operations are performed by taking the null values in the second input matrix into account. This tends to more computation overhead and more time for multiplication. FASTsparseMUL [14] and RANGEsparseMUL [15] use their own sparse matrix input format or layout effectively. It leads to better performance in terms of execution time and scalability compared to HAMA based iterative approaches as well as VLCA approach. But by using a range based sparse matrix compact representation format, Big-RANGEsparseMUL (Extended RANGEsparseMUL Algorithm which implements Big-Range Sparse Compact model) performs a bit better than FASTsparseMUL.
The algorithm Big-RANGEsparseMUL removes the drawbacks of existing big data approaches in sparse matrix multiplication. It concentrates on reducing the multiplication time and increasing the scalability in the scenario of sparse Big Data. It uses the compact representation techniques of sparse data, converts them into a mapreducible format and performs a sparse matrix multiplication with less execution time. The results show that the algorithm performs better when compared to big sparse matrix multiplication using HAMA_Hadoop, HAMA_HPMR[9-10], VLCA [13] and FASTsparseMUL [14]. It can be implemented in many big sparse data applications.

2. Problem Statement

Usually, sparse data comprise a large number of missing values which are useless in the decision making process. If the original sparse data is stored in the main memory, much of the memory goes waste by missing values. So at the preprocessing stage, it must be converted into a compact form to save memory space and to make the computation process easy. But the compact sparse data representations must be converted into a mapreducible format to make them suitable for big sparse data applications and efficient big data algorithms must be implemented to get faster results with optimal scalability.

3. Problem Solving and Innovative Content

Figure 1 depicts the row representation of a sparse matrix. The three sets of adjacent non-null values are represented as ranges. Though this compact sparse data representation solve the storage and retrieval problems to the maximum extent, the matrix multiplication problem is yet to be solved in the big data scenario.

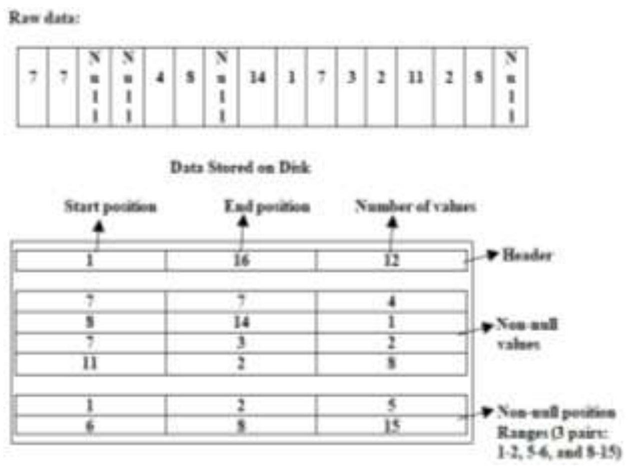


Fig. 1: A compact sparse row representation with positions represented as ranges

We applied a big data-programming model called mapreduce here to solve the problem. Figure 2 exposes the control flow of a typical mapreduce job. We may give a huge amount of data as input. At first, mapreduce partitions the input. Each mapper node receives a partition where map task starts its execution. Mapreduce job processes a fixed number of mapper nodes in parallel. Mapreduce job collects the outputs from all the mappers. Each reducer receives this output collection. Every mapreduce job follows split, sort, and merge operation sequence. At last, mapreduce fetches the outputs from all reducer functions and accumulates as one final output file.

A solution to a big data problem should fulfil the three prerequisites mentioned below.

The input data must be distributable in nature.

All the local outputs must lead to the global/ final output.

It should feasible for the implementation of the mapreduce model.

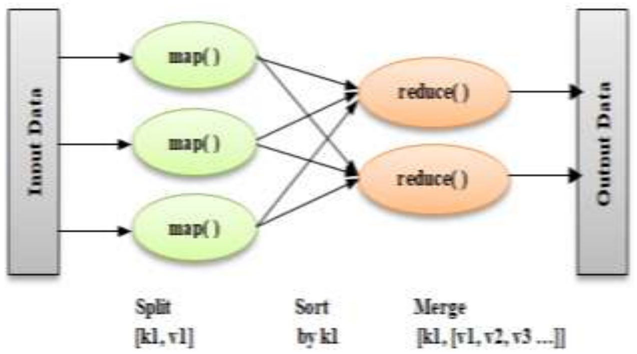


Fig. 2: Mapreduce flow.

Initially we used two separate files to store the mapreducible formats of sparse matrices. Later we further simplified the problem, by generating a single input _le from the concatenation of separate files.The sample mapreducible format of a sparse matrix row is shown in Figure 3.

The ranges having exactly two adjacent elements are not storedwith the symbol R. Instead these elements are just mentioned with their non-null positions to save storage space and to reduce data access time. For example, in Figure 3, row #1 of matrix M startswith a range of exactly two adjacent elements (i.e., 7,7 respectively). This range is not preceded by the symbol R.

The proposed Big-RANGEsparseMUL uses mapreduce to accomplish the goal. Its sub-algorithm RANGE_MR_sparseMUL implements mapreduce. For time being, the elaboration of the subalgorithm RANGE_MR_sparseMUL is skipped. However, Its map and reduce tasks are well explained by us earlier[15]. The pseudo code of Big-RANGEsparseMUL is shown below.

The Hadoop 2.6.0 is installed in pseudo distributed mode and experiments are conducted to use the proposed compact representation model effectively. The results in the following section show the better performance of Big-RANGEsparseMUL compared to VLCA, FASTsparseMUL, and the matrix multiplication approaches using the recent tools based on distributed framework HAMA[9-10] in terms of execution time and scalability.

Algorithm Big-Range_Sparse_Compact(Matrix A, Matrix B)

Input: The original matrices A and B;
File A consists of the original sparse matrix of size m*n.
File B consists of the original sparse matrix of size n*k.
Output: The target data file D; // File D consists of the compact forms of the original sparse matrices.

Algorithm Big-Range_Sparse_Compact(Matrix A, Matrix B)
Input: The original matrices A and B;
File A consists of the original sparse matrix of size m*n.
File B consists of the original sparse matrix of size n*k.
Output: The target data file D; // File D consists of the compact forms of the original sparse matrices.

Algorithm Big-Range_Sparse_Compact(Matrix A, Matrix B)

Input: The original matrices A and B;
File A consists of the original sparse matrix of size m*n.
File B consists of the original sparse matrix of size n*k.
Output: The target data file D; // File D consists of the
compact
forms of the original sparse matrices.

Algorithm Big-Range_Sparse_Compact(Matrix A, Matrix B)

Input: The original matrices A and B;
File A consists of the original sparse matrix of size m*n.
File B consists of the original sparse matrix of size n*k.
Output: The target data file D; // File D consists of the
compact

Algorithm Big-Range_Sparse_Compact(Matrix A, Matrix B)

Input: The original matrices A and B;
File A consists of the original sparse matrix of size m*n.
File B consists of the original sparse matrix of size n*k.
Output: The target data file D; // File D consists of the
compact

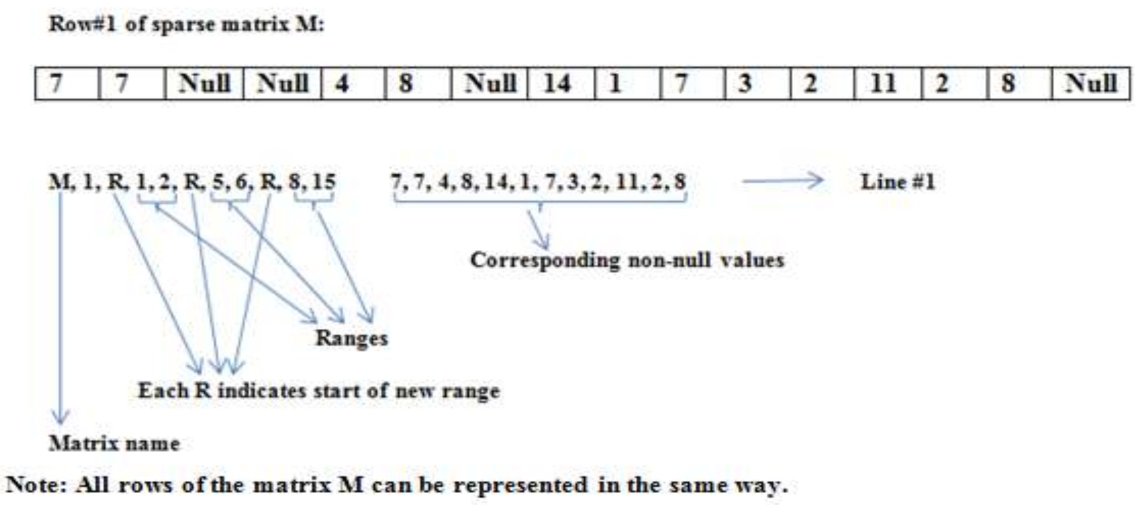

Row#1 of sparse matrix M:
7 | 7 | Null | Null | 4 | 8 | Null | 14 | 1 | 7 | 3 | 2 | 11 | 2 | 8 | Null
M, 1, R, 1, 2, R, 5, 6, R, 8, 15
7, 7, 4, 8, 14, 1, 7, 3, 2, 11, 2, 8
Line #1
Corresponding non-null values
Ranges
Each R indicates start of new range
Matrix name
Note: All rows of the matrix M can be represented in the same way.

Fig. 3: The Big-Range Sparse Compact model : A row of the sparse matrix M, represented as a line in the input file.

```
// Algorithm  Big-Range_Sparse_Compact continuation
1: for  i = 0..m do
2:   str1="";     // create  two empty strings.
3:   str2="";
5:   for j = 0..n do
6:     v=w=0;
7:      w=j;
8 :    if A[i][j] !=Null then
// Avoiding storage of null values of matrix A
9:        str1+="A,i";
10:       while A[i][j] != Null do
11:         v=j;
12:         str2+=","+A[i][j];
13:         j++;
14:       end while
15:       if (v-w+1)>1 then
16:        str1+=",R"+","+w+","+v;
17:       end if
18:       if (v-w+1)<1 then
19:        str1+=","+w;
20:       end if
21:       if (v-w+1)=1 then
22:         str1+=","+w+","+v;
23:       end if
24 :    end if
25:  end for
26:   line=str1+"\t"+str2;
27:  Write line to file f1;
// Writing each row of matrix A as a line in file f1
28:end for
29: for  i = 0..n do
30:    str1="";    // create  two empty strings.
31:    str2="";
32:   for j = 0..k do
33 :     v=w=0 ;
34:      w=j;
35 :    if B[i][j] !=Null then
// Avoiding storage of  null values of matrix B
36:         str1+="B,i";
37:       while B[i][j] != Null do
38:         v=j;
39:         str2+=","+B[i][j];
40:         j++;
41:       end while
42:       if (v-w+1)>1 then
43:         str1+=",R"+","+w+","+v;
44:       end if
```

```
// Algorithm  Big-Range_Sparse_Compact continuation
45:       if (v-w+1)<1then
46:        str1+=","+w;
47:       end if
48:       if (v-w+1)=1 then
49:         str1+=","+w+","+v;
50:       end if
51 :    end if
52:   end for
53:   line=str1+"\t"+str2;
54:  Write line to file f2;
       // Writing each row of matrix B, as a line in
          file f2
55:end for
56: Concatenate f1 and f2 to get file D.
// Collective compact  representation of both matrices
   A and B.
```

4. Results and Comparison

Table 1 shows comparative analysis of Big-RANGEsparseMUL with existing matrix multiplication approaches in the big sparse data scenario. Big-RANGEsparseMUL is executed on single node Hadoop-pseudo distributed cluster environment with 1 % sparse matrices having dimensions varying from 32 to 320.

Big-RANGEsparseMUL makes best use of the proposed Big-Range Sparse Compact model and shows approximately 3.4 times, 3.1 times, 2.0 times, and 1.2 times reduction in time complexity on average compared to the sparse matrix multiplication using HAMA_Hadoop, HAMA_HPMR, VLCA, and FASTsparseMUL respectively. The execution times of different matrix multiplication approaches are tabulated and compared in Table 2, Figure 4 respectively. The snapshot of a sample input file is shown in Figure 5. The resultant overviews for the mapreduce job of Big-RANGEsparseMUL are shown in Figure 6 and Figure 7 respectively.

is used for the calculation of scale up values.

$$\text{scaleup}(\text{dimension}) = \log(T(\text{dimension})/T(32)) \qquad (1)$$

(Here T denotes the time for execution).

Table 1: Analytical Comparison of Big-RANGESparseMUL with Various Matrix Multiplication Approaches in the Big Sparse Data Scenario

Approach/ Algorithm	Advantages	Limitations
HAMA based iterative approach (Mapreduce)	No constraint on problem size	□ Takes multiple rounds for matrix multiplication
VLCA(Map-reduce)	□ No constraint on problem size	□ No preprocessing to remove null values in the input sparse matrices.
	□ Reduction in execution time	□ No use of any special format for input sparse matrices.
	□ Takes single mapreduce job	□ No focus on null values in second input matrix
		□ Unnecessary computation overhead which includes null values of second input matrix.
FASTsparseMUL (MapReduce)	□ No constraint on problem size	□ Preprocessing overhead
	□ Maximum reduction in execution time	□ Mainly intended for sparse matrix multiplication.
	□ Takes single mapreduce job.	□ Application of the algorithm to dense matrices is yet to be studied.
	□ Shows maximum scalability	
	□ Makes best use of a special format for input sparse matrices.	
Big-RANGEsparseMUL(Mapreduce)	□ No constraint on problem size	□ Preprocessing overhead
	□ Takes single mapreduce job.	□ Mainly intended for sparse matrix multiplication.
	□ Makes best use of a special format for input sparse matrices.	□ Application of the algorithm to dense matrices is yet to be studied.
	□ Works best when the input sparse matrices have large number of sets of adjacent non-null values.	
	□ Shows better reduction in execution time and improvement in scalability than FASTsparseMUL.	
	□ Overcomes the drawbacks of state-of-the-art matrix multiplication approaches in Big sparse data scenario.	

Scale up is inversely proportional to the scalability. The scalability improvement of Big-RANGEsparseMUL compared to sparse matrix multiplication using HAMA_Hadoop, HAMA_HPMR, VLCA and FASTsparseMUL is depicted in Figure 8 and tabulated in Table 3. Since scaleup values of FASTsparseMUL are close to that of Big-RANGEsparseMUL, three decimal places are taken after the decimal point of each scaleup value in both approaches for better comparison. After the initial execution of mapreduce job

Table 2: Execution Times of Various Matrix Multiplication Approaches

	Execution time (sec)				
Matrix dimension	HAMA_ Hadoop	HAMA_ HPMR	VLCA	FASTsparseMUL	Big-RANGEsparseMUL
32	16	16	12	41	36
64	85	71	48	37	32
128	102	101	69	35	29
192	131	115	79	42	35
256	181	172	103	47	40
320	228	202	125	52	44

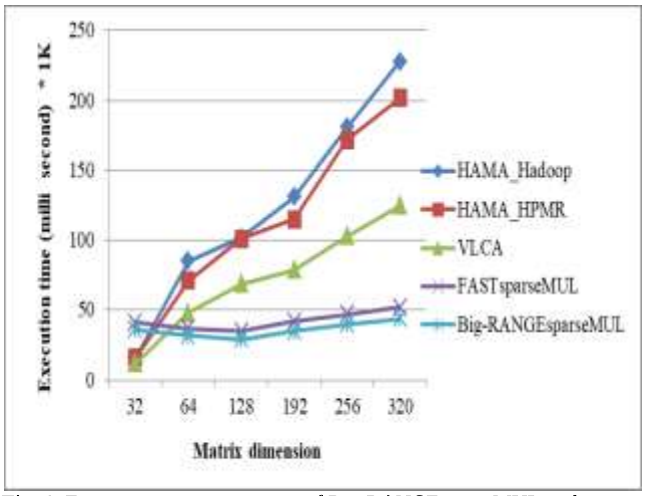
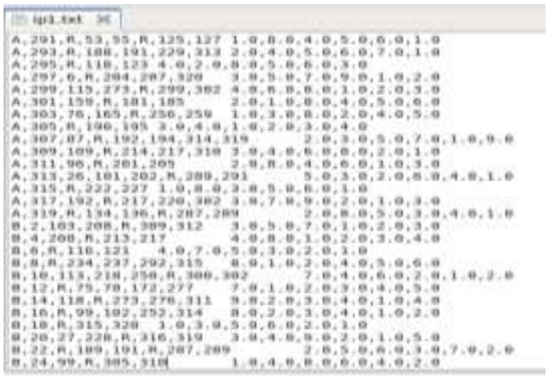


Fig. 4: Execution time comparison of Big-RANGEsparseMUL with various approaches



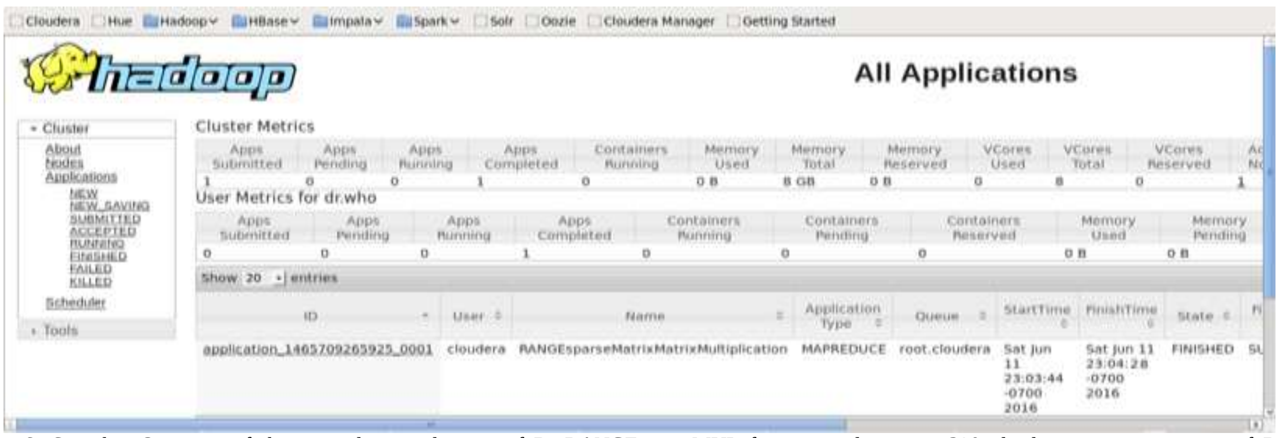
```
A,291,R,53,55,R,125,127 1.0,0.0,4.0,5.0,6.0,1.0
A,293,R,188,191,229,313 2.0,4.0,5.0,6.0,7.0,1.0
A,295,R,118,123 4.0,2.0,8.0,5.0,6.0,3.0
A,297,6,R,204,207,320    3.0,5.0,7.0,9.0,1.0,2.0
A,299,115,273,R,299,302 4.0,8.0,6.0,1.0,2.0,3.0
A,301,159,R,181,185      2.0,1.0,8.0,4.0,5.0,6.0
A,303,76,165,R,256,259   1.0,3.0,8.0,2.0,4.0,5.0
A,305,R,190,195 3.0,4.0,1.0,2.0,3.0,4.0
A,307,87,R,192,194,314,319        2.0,3.0,5.0,7.0,1.0,9.0
A,309,109,R,214,217,318 3.0,4.0,6.0,8.0,2.0,1.0
A,311,96,R,201,205       2.0,8.0,4.0,6.0,1.0,3.0
A,313,26,101,202,R,289,291        5.0,3.0,2.0,6.0,4.0,1.0
A,315,R,222,227 1.0,8.0,3.0,5.0,6.0,1.0
A,317,192,R,217,220,302 3.0,7.0,9.0,2.0,1.0,3.0
A,319,R,134,136,R,287,289         2.0,8.0,5.0,3.0,4.0,1.0
B,2,103,208,R,309,312    3.0,5.0,7.0,1.0,2.0,3.0
B,4,208,R,213,217        4.0,8.0,1.0,2.0,3.0,4.0
B,6,R,118,121    4.0,7.0,5.0,3.0,2.0,1.0
B,8,R,234,237,292,315    8.0,1.0,2.0,4.0,5.0,6.0
B,10,113,218,250,R,300,302        7.0,4.0,6.0,2.0,1.0,2.0
B,12,R,75,78,172,277     7.0,1.0,2.0,3.0,4.0,5.0
B,14,118,R,273,278,311   9.0,2.0,3.0,4.0,1.0,4.0
B,16,R,99,102,252,314    8.0,2.0,3.0,4.0,1.0,2.0
B,18,R,315,320  1.0,3.0,5.0,6.0,2.0,1.0
B,20,27,228,R,316,319    3.0,4.0,9.0,2.0,1.0,5.0
B,22,R,189,191,R,287,289          2.0,5.0,8.0,3.0,7.0,2.0
B,24,99,R,305,310        1.0,4.0,8.0,6.0,4.0,2.0
```

Fig. 5: Snapshot of part of the sample input file for the matrix dimension 320

RANGE_MR_sparseMUL for matrix dimension 32, the next two mapreduce jobs exhibited negative scaleup values. The reason is, after the initial mapreduce job execution, the required mapreduce program is loaded in cache memory. So the next immediate jobs take lesser time compared to initial mapreduce job leading to negative scaleup values. However this effect is nullified for later mapreduce jobs due to the use of large input files having matrix dimensions 192, 256 and 320.

5. Conclusion

An efficient compact matrix representation to deal with massive sparse data is proposed and supported with a big data approach on sparse matrix multiplication in this paper. Many big sparse data applications such as genomics applications, analysis of protein mass-spectrometry data, sparse brain imaging applications, sequential testing approaches, algorithmic aspects of sparse recovery, learning sparse latent models and so on may use this algorithm. In future our focus would be on executing the model in Spark and R-Hadoop environments.



All Applications

Cluster Metrics

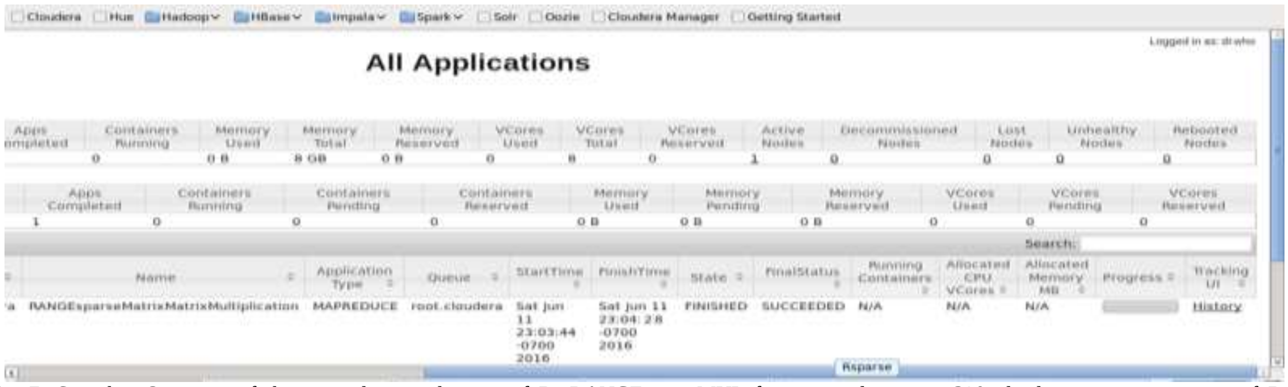
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Ac No
1	0	0	1	0	0 B	8 GB	0 B	0	8	0	1

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending
0	0	0	1	0	0	0	0 B	0 B

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	Fi
application_1465709265925_0001	cloudera	RANGEsparseMatrixMatrixMultiplication	MAPREDUCE	root.cloudera	Sat Jun 11 23:03:44 -0700 2016	Sat Jun 11 23:04:28 -0700 2016	FINISHED	SU

Fig. 6: Snapshot Overview of the mapreduce application of BigRANGEsparseMUL for matrix dimension 320, displaying execution time of Big-RANGEsparseMUL for the matrix dimension 320.(Finish Time - Start Time = 23:04:28 -23:03:44 = 44 sec (shown in Table 2)) – PART-A



All Applications

Apps ompleted	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
	0	0 B	8 GB	0 B	0	8	0	1	0	0	0	0

Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
1	0	0	0	0 B	0 B	0 B	0	0	0

Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Progress	Tracking UI
RANGEsparseMatrixMatrixMultiplication	MAPREDUCE	root.cloudera	Sat Jun 11 23:03:44 -0700 2016	Sat Jun 11 23:04:28 -0700 2016	FINISHED	SUCCEEDED	N/A	N/A	N/A		History

Fig. 7: Snapshot Overview of the mapreduce application of BigRANGEsparseMUL for matrix dimension 320, displaying execution time of Big-RANGEsparseMUL for the matrix dimension 320.(Finish Time - Start Time = 23:04:28 -23:03:44 = 44 sec (shown in Table 2)) – PART-B

Table 3: Scaleup values of Various Matrix Multiplication Approaches

Scale up					
Matrix dimension	**HAMA_ Hadoop**	**HAMA_ HPMR**	**VLCA**	**FASTsparseMUL**	**Big-RANGEsparseMUL**
32	0	0	0	0	0
64	0.73	0.65	0.6	-0.045	-0.051
128	0.8	0.8	0.76	-0.069	-0.094
192	0.91	0.86	0.82	0.01	-0.012
256	1.05	1.03	0.93	0.059	0.046
320	1.15	1.1	1.02	0.103	0.087

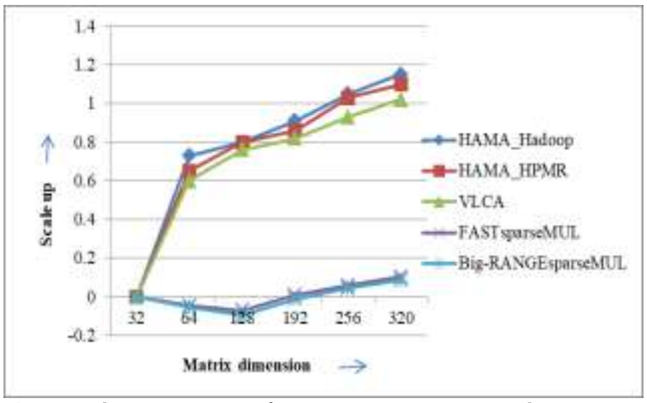


Fig. 8: Scaleup comparison of Big-RANGEsparseMUL with various approaches

References

[1] Abadi DJ (2007), Column-stores for wide and sparse data, Proceedings of the third Biennial conference on Innovative Data Sys temsResearch(CIDR), 1-6. Neelima B & Prakash SR (2012), Effective Sparse Matrix Repres- entation for the GPU Architectures. International Journal of Computer Science, Engineering and Applications (IJCSEA) 2, 151-165. Yuster R & Zwick U (2005), Fast sparse matrix multiplication, Journal of ACM Transactions on Algorithms (TALG) 1, 2-13.

[2] Dean J & Ghemawat S (2008), MapReduce: simplified data processing on large clusters. Communications of the ACM 51, 107-113.

[3] White T (2009), Hadoop: The Definitive Guide, 4th edn. O'Reilly Media, Inc., CA, pp. 3-337.

[4] Buluc A & Gilbert JR (2011), Parallel Sparse Matrix-Matrix multiplication and Indexing: Implementation and experiments. SIAM Journal on Scientific Computing 34, C170-C191.

[5] Ballard G, Bulluc A, Demmel J, Grigori L, Lipshitz B, Schwartz & Toledo S (2013), Communication Optimal Parallel Multiplication of Sparse Random Matrices, Proceedings of the 25th Annual ACM symposium on Parallelism in Algorithms and Architectures, 222-231, https://doi.org/10.1145/2486159.2486196

[6] Smith S, Ravindran N, Sidiropoulos ND & Karypis G (2015), SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplica-

[7] tion, Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium, 1058-1067, https://doi.org-/10.1109/IPDPS.2015.27

[8] Seo S, Yoon EJ, Kim J, Jin S, Kim JS & Maeng S (2010), HAMAAn Efficient Matrix Computation with the MapReduce Framew-ork, Proceedings of the IEEE 2nd International Conference onCloud ComputingTechnology and Science(CloudCom), 721-726, https://doi.org/10.1109/CloudCom.2010.17

[9] Seo S, Jang I, Woo K, Kim I, Kim JS & Maeng S (2009), HPMR: Prefetching and Pre-shuffling in Shared MapReduce Computation
[10] Environment, Proceedings of the 11th IEEE International Conference on Cluster Computing, 1-8, https://doi.org/10.1109/CLUSTR.2009.5289171
[11] Myung J & Lee S (2012), Matrix Chain multiplication via multi-way join algorithms in mapreduce, Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, Article No. 53, https://doi.org/10.1145/2184751.2184817
[12] Ceccarello M & Silvestri F (2015), Experimental Evaluation of Multi Round Matrix Multiplication on MapReduce, Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX '15), 119-132, https://doi.org/10.1137/1.9781611973754.11
[13] Zheng JH, Zhang LJ , Zhu R, Ning K & Liu D (2013), Parall-el Matrix Multiplication Algorithm Based on Vector Linear Combination Using MapReduce, Proceedings of the IEEE 9th World-Congress on Services, 193-200, https://doi.org/10.1109/SERVICES.2013.67
[14] Somasekhar G & Karthikeyan K (2017), Fast Matrix Multiplication with Big Sparse Data. Cybernetics and Information Technologies 17, 16-30.
[15] Somasekhar G & Karthikeyan K (2017), The Range based Mapreduce Algorithm for Large Sparse Matrix Multiplication, Proceedings of the International Conference on Big Data and Cloud Computing, 73-80.