EasyChair Preprint

№ 10818

MapReduce: Big Data Maintained Algorithm

Deepak Uprety, Dyuti Banarjee, Nitish Kumar and
Abhimanyu Dhiman

September 1, 2023

MapReduce: Big Data Maintained Algorithm

Dr. Deepak Chandra Uprety (Asso. Prof., CSE) [1], Dr. Dyuti Banarjee (Asst. Prof., AI & ML) [2], Nitish Kumar (Asst. Prof., CSE) [3], Abhimanyu Dhiman (Asst. Prof., CSE) [4]

[1] IEC University, Baddi, Himanchal Pradesh, India,
deepak.glb@gmail.com,
[2] Koneru Lakshmaiah Education Foundation, Green Fields, Vaddeswaram, Andhra Pradesh-522302, India
dyuti738900@gmail.com
[3] LPU University, Jalandhar - Delhi, Grand Trunk Rd, Phagwara, Punjab 144001, India,
nitishchauhan946@gmail.com,
[4] IEC University, Baddi, Himanchal Pradesh, India,
abhimanyu.dhiman1995@gmail.com

Abstract

Big Data Maintained Algorithm is the practice of probing large amounts of data. The challenges include capturing, scrutiny, storage, searching, sharing, exposure, transferring and privacy violations. MapReduce is an uncomplicated, scalable and fault-tolerant data processing framework that enables us to process a substantial amount of data. MapReduce framework has generated a lot of concern in a wide range of areas. It currently is an elegant model for data severe applications due to the easy interface of programming, high scalability and fault tolerance capability. It leads to a new gesture of awareness to large-scale algorithms for data analysis. Algorithms cannot handle the enormous data well, for improving the time competence, for this MapReduce is best solution. So, this paper indicates the MapReduce applications in optimization algorithms and also provides a discussion of differences between its applications as well as some guidelines for future research. MapReduce is presently the most admired programming model for big data processing, and Hadoop is a well-known MapReduce accomplishment platform. To resolve these problems, in this paper, we propose MapReduce programming model for handing out huge datasets in distributed systems; it helps programmers to write programs that process big data. The goal of this paper is to examine MapReduce research trends, and current research efforts for enhancing MapReduce performance and capabilities. This Study accomplished that the research directions of MapReduce concerned with either enhancing MapReduce programming model or adopting MapReduce for deploying accessible algorithm to run with MapReduce programming model.

Keywords

Big data, distributed computing, MapReduce Application Programming Interface, scalability and MapReduce programming model

1. MapReduce Algorithm

MapReduce is an encoding prototype within the Hadoop frame that is used to access big data stored in the Hadoop File System (HDFS). It is a middle part component, necessary to the presentation of the Hadoop frame.

MapReduce facilitates concurrent handing out by splitting petabytes of data into minor chunks, and processing them in parallel on Hadoop service servers. In the end, it aggregates all the data from frequent servers to go back a consolidated output back to the request.

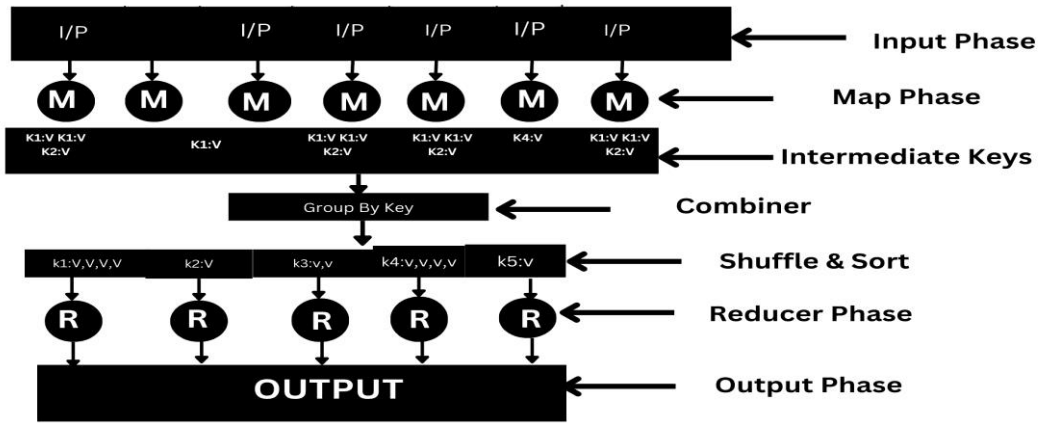

Fig.1 MapReduce Algorithm

Input Phase

It is an evidence reader that sends data in the form of key-value pairs and transforms every input file evidence to the mapper.

Map Phase

It is a client distinct job. It generates zero or more key-value pairs with the help of a series of key-value pairs and processes each of them.

Intermediate Keys

Mapper generated key-value pairs are called as midway keys.

Combiner

Combiner takes mapper interior keys as input and applies a user-defined strategy to unite the values in a small extent of one mapper.

Shuffle and Sort

Shuffle and Sort is the first step of reducer job. When reducer is running, it downloads all the key-value pairs onto the neighboring machine. Each key -value pairs are stored by key into a bigger data list. This data list groups the parallel keys in concert so that their principles can be iterated easily in the reducer job.

Reducer phase

This section gives zero or supplementary key-value pairs after the data can be shared, filtered and aggregated in a number of ethnicity and it requires a large range of special consideration.

Output phase

It has an output formatter, from the reducer function and writes them onto a file using a record writer that translates the last key-value pairs. A Hadoop cluster with 20,000 contemptible product servers and 256 Mb blocks of data in each, can route around 5 Tb of data at the same time. This reduces the allowance time as compared to sequential allowance of such a large data set.

MapReduce was once the only way through which the data stored in the HDFS could be retrieved, but today, there are other query-based systems such as Hive and Pig that are used to get back data from the HDFS using SQL-like statements.

2. MapReduce API (Application Programming Interface)

Programming in MapReduce

Curriculum and methods are alarmed in the operations of MapReduce encoding. For this, we center on the following concepts.

(i) Job context interface
(ii) Job class
(iii) Mapper class
(iv) Reducer class

(i) Job context interface

The job context sub interfaces are:

(a) Map context

It defines the framework which is given to the mapper:
Mapcontext< KEYIN, VALUEIN, KEYOUT, VALUEOUT >

(b) Reduce context

It defines the framework which is approved to reducer:
Reducecontext< KEYIN, VALUEIN, KEYOUT, VALUEOUT >

The Main division of job context interface is a job class that helps with execution.

(ii) Job class

The significant class in the mapreduce API is job class. The Job class allows the client to job put together, capitulation, implementation and the doubt state. Until the submitted job the set methods work, after that they will chuck an unlawful state exemption.

Methods of job class

(a)	**getjobName( )**:	job name specified by the user
(b)	**getjobState( )**:	Returns the job current state
(c)	**isComplete ( )**:	Checks whether the job is finished or not
(d)	**setInputFormatClass( )**:	Sets the input format for the job
(e)	**setjobName(String name)**:	Sets the job name specified by the user
(f)	**setOutputFormatClass( )**:	Sets the output format for the job
(g)	**setMapperClass(Class)**:	Sets the mapper for the job
(h)	**setReducerClass(Class)**:	Sets the reducer for the job
(i)	**setPartitionerClass(Class):**	Sets the partitioner for the job
(j)	**setCombinerClass(Class):**	Sets the combiner for the job.

(iii) Mapper class

It defines a map job, it maps input key or value to a collection of intermediate key or value pairs. Maps are individual task that take input records to intermediate records. It maps zero or more output pairs from giving an input pair.

The most significant method of mapper class is map. The syntax is map (KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Mapper.Context context)

(iv) Reducer class

It defines the reducer job in mapreduce. Reduces is a group of middle values, that share a key to a smaller set of values via JobContext.getConfiguration () method. We can access the configuration for a job. Three phases of reducers are

(a) Shuffle
The sorted output of reducer copies from every mapper using http across the network.

(b) Sort
When the outputs are fetched, both the phases (shuffle and sort) occur at a time and they compound the data.

(c) Reduce
Syntax of this phase is reduce (Object, Iterable, Context).

The most important method of reducer class is reduce. The syntax is reduce (KEYIN key, Iterable<VALUEIN> values, org.apache.hadoop.mapreduce)

3. HOW MAPREDUCE WORKS

Map and Reduce

At the root of MapReduce, there are two functions, Map and Reduce. They are sequenced one after the other.

- **Map**

  The Map function takes input from the disk as <key, value> pairs, processes them, and produces another set of intermediate <key, value> pairs as output.

  For example, the input data is first split into minor blocks. Each block is then assigned to a mapper for processing. For example, if a file has 100 records to be processed, 100 mappers can run together to process one record each or maybe 50 mappers can run together to process two records each. The Hadoop framework decides how many mappers to use based on the size of the data to be processed and the memory block available on each mapper server.

- **Reduce**

  The Reduce function also takes inputs as <key, value> pairs, and produces <key, value> pairs as output. The types of keys and values are dissimilar based on the use case. All inputs and outputs are stored in the HDFS. While the map is a necessary step to filter and sort the initial data, the reduce function is elective.

  <k1, v1> -> Map () -> list (<k2, v2>)

  <k2, list (v2)> -> Reduce () -> list (<k3, v3>)

  After all the mappers complete processing, the framework shuffles and sorts the results before passing them on to the reducers. A reducer cannot start while a mapper is still in progress. All the map output values that have the same key are assigned to a single reducer, which then aggregates the values for that key.

  Mappers and Reducers are the Hadoop servers that run the Map and Reduce functions correspondingly. It doesn't issue if these are the same or different servers.

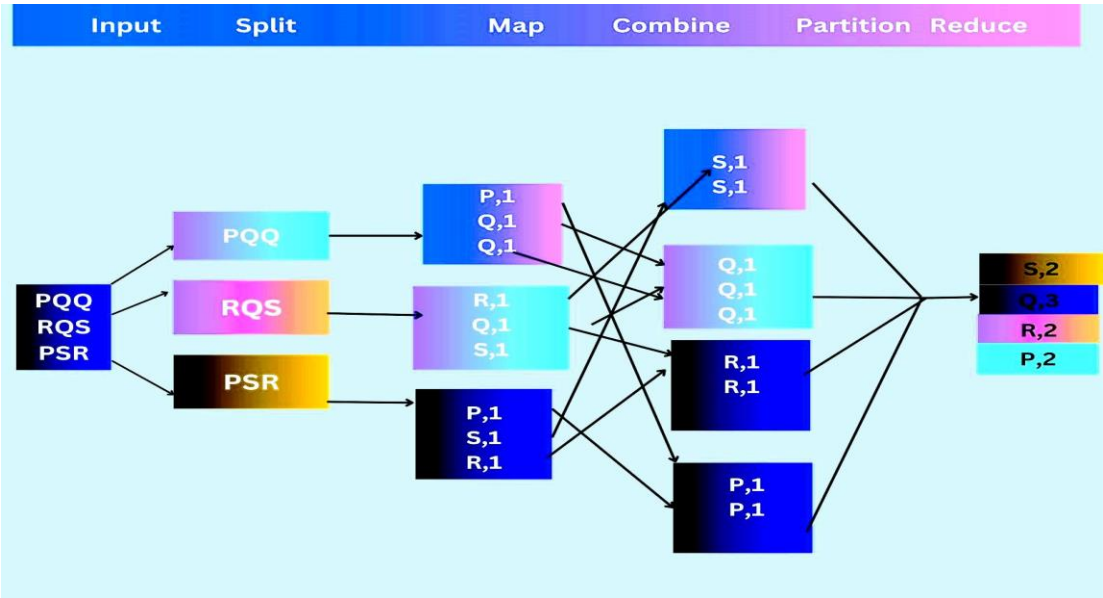

Fig.3 MapReduce Working style

Combine and Partition

There are two intermediate steps between Map and Reduce.

- **Combine**

It is an elective process. The combiner is a reducer that runs independently on each mapper server. This makes shuffling and sorting easier as there is less data to work with. Often, the combiner class is set to the reducer class itself, due to the growing and associative functions in the reduce function. However, if needed, the combiner can be a divide class as well.

- **Partition**

It is the process that translates the <key, value> pairs resulting from mappers to another set of <key, value> pairs to feed into the reducer. It decides how the data has to be accessible to the reducer and also assigns it to a particular reducer. There are as many partitions as there are reducers. So, once the partitioning is complete, the data from each partition is sent to a specific reducer.

4. MapReduce Partitionar

Partitioner in MapReduce

Intermediate-outputs in the key-value pairs partitioned by a partitioner. The number of reducer tasks is equal to the number of partitions in the job.

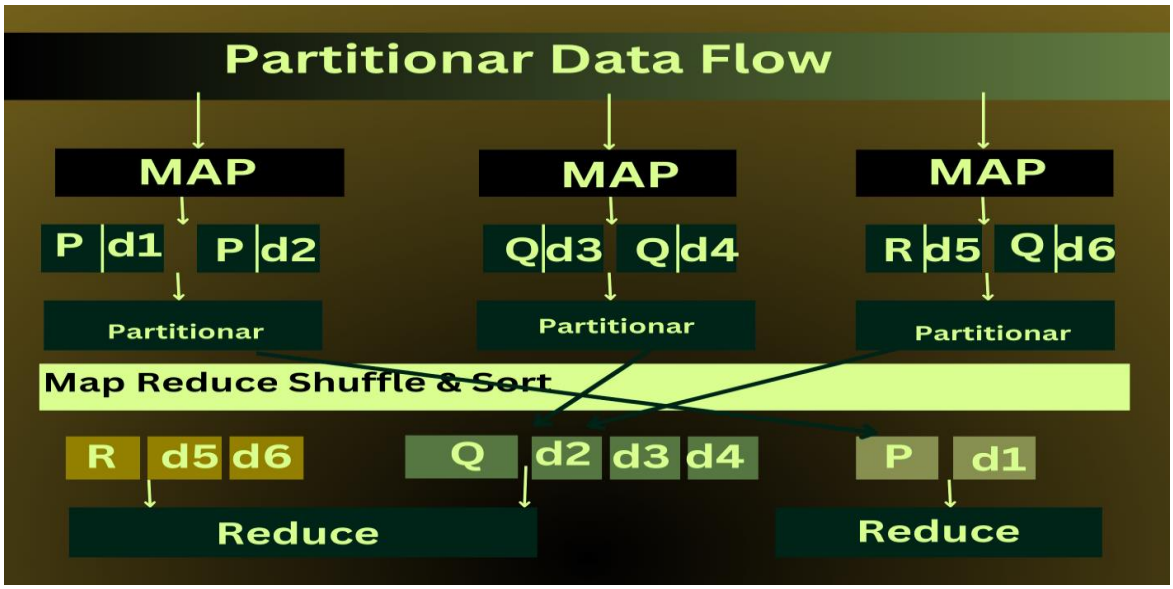

Fig.4 MapReduce Partitionar

5. A MapReduce Example

Consider an ecommerce system that receives a million requests every day to process payments. There may be several exceptions thrown during these requests such as “payment declined by a payment gateway," "out of inventory," and "invalid address."A developer wants to analyze last four days' logs to understand which exception is thrown how many times.

MapReduce is an apt programming model. Multiple mappers can process these logs concurrently: one mapper could process a day's log or a subset of it based on the log size and the memory block available for processing in the mapper server.

Map

For simplification, let's assume that the Hadoop framework runs just four mappers. Mapper 1, Mapper 2, Mapper 3, and Mapper 4. The value input to the mapper is one record of the log file. The key could be a textstring such as "file name + line number." The mapper, then, processes each record of the log file to produce key value pairs. Here, we will just use filler for the value as '1.'The outputs from the mappers look like this:

Mapper 1 -> <Exception P, 1>, <Exception Q, 1>, <Exception P, 1>, <Exception R, 1>, <Exception P, 1>
Mapper 2 -> <Exception Q, 1>, <Exception Q, 1>, <Exception P, 1>, <Exception P, 1>
Mapper 3 -> <Exception P, 1>, <Exception R, 1>, <Exception P, 1>, <Exception Q, 1>, <Exception P, 1>
Mapper 4 -> <Exception Q, 1>, <Exception R, 1>, <Exception R, 1>, <Exception P, 1>

Combine

Assuming that there is a combiner running on each mapper—Combiner 1 …Combiner 4—that calculates the count of each exception, which is the same function as the reducer.

The output of Combiners will be:

Combiner 1: <Exception P, 3>, <Exception Q, 1>, <Exception R, 1>
Combiner 2: <Exception P, 2> <Exception Q, 2>
Combiner 3: <Exception P, 3> <Exception Q, 1> <Exception R, 1>
Combiner 4: <Exception P, 1> <Exception Q, 1> <Exception R, 2>

Partition

After this, the partitioner allocates the data from the combiners to the reducers. The data is also sorted for the reducer. The input to the reducers will be as below:

Reducer 1: <Exception P> {3, 2, 3, 1}
Reducer 2: <Exception Q> {1, 2, 1, 1}
Reducer 3: <Exception R> {1, 1, 2}

If there were no combiners involved, the input to the reducers will be as below:

Reducer 1: <Exception P> {1, 1, 1, 1, 1, 1, 1, 1, 1}
Reducer 2: <Exception Q> {1, 1, 1, 1, 1}
Reducer 3: <Exception R> {1, 1, 1, 1}

Here, the example is a simple one, but when there are terabytes of data involved, the combiner process' improvement to the bandwidth is significant.

Reduce

Now, each reducer just calculates the total count of the exceptions as:

Reducer 1: <Exception P, 9>
Reducer 2: <Exception Q, 5>
Reducer 3: <Exception R, 4>

The data shows that Exception A is frightened more often than others and requires more attention. When there are more than a few weeks' or months' of data to be processed together, the potential of the MapReduce program can be truly demoralized.

References

[1] Aarthee, S., & Prabakaran, R. (2023). Energy-Aware Heuristic Scheduling Using Bin Packing MapReduce Scheduler for Heterogeneous Workloads Performance in Big Data. *Arabian Journal for Science and Engineering*, *48*(2), 1891-1905.

[2] Yang, S., Jin, W., Yu, Y., & Hashim, K. F. (2023). Optimized hadoop map reduce system for strong analytics of cloud big product data on amazon web service. *Information Processing & Management*, *60*(3), 103271.

[3] Bawankule, K. L., Dewang, R. K., & Singh, A. K. (2023). Early straggler tasks detection by recurrent neural network in a heterogeneous environment. *Applied Intelligence*, *53*(7), 7369-7389.

[4] Kalia, K., & Gupta, N. (2021). Analysis of hadoop MapReduce scheduling in heterogeneous environment. *Ain Shams Engineering Journal*, *12*(1), 1101-1110.

[5] Luo, C., Cao, Q., Li, T., Chen, H., & Wang, S. (2023). MapReduce accelerated attribute reduction based on neighborhood entropy with Apache Spark. *Expert Systems with Applications*, *211*, 118554.

[6] Pandey, R., & Silakari, S. (2023). Investigations on optimizing performance of the distributed computing in heterogeneous environment using machine learning technique for large scale
data set. *Materials Today: Proceedings*, *80*, 2976-2982.

[7] Jagadish Kumar, N., & Balasubramanian, C. (2023). Hybrid Gradient Descent Golden Eagle Optimization (HGDGEO) Algorithm-Based Efficient Heterogeneous Resource Scheduling for Big Data Processing on Clouds. *Wireless Personal Communications*, *129*(2), 1175-1195.

[8] Slagter, K., Hsu, C. H., Chung, Y. C., & Zhang, D. (2013). An improved partitioning mechanism for optimizing massive data analysis using MapReduce. *The Journal of Supercomputing*, *66*, 539-555.

[9] Che, D., Safran, M., & Peng, Z. (2013). From big data to big data mining: challenges, issues, and opportunities. In

Database Systems for Advanced Applications: 18th International Conference, DASFAA 2013, International Workshops: BDMA, SNSM, SeCoP, Wuhan, China, April 22-25, 2013. Proceedings 18 (pp. 1-15). Springer Berlin Heidelberg.
[10] Uma Maheswara Rao, S., & Lakshmanan, L. (2023). Security and scalability issues in big data analytics in heterogeneous networks. *Soft Computing*, 1-7.
[11] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proc. the 6th conference on Symposium on Operating Systems Design & Implementation*, 2004, pp. 137-150.
[12] T. White, *Hadoop: The Definitive guide*, 1st ed.: O'Reilly Media, 2010.
[13] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *Proc. the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 29-43.
[14] A. Rajaraman and J. Ullman, *Mining of Massive Datasets*, Cambridge University Press, 2011, ch. 2.
[15] A. M. Middleton, *Data-Intensive Technologies for Cloud Computing.* B. Furht and A. Escalante, *Handbook of Cloud Computing*, New York: Springer, 2010, ch. 5.
[16] S. Chen and S. W. Schlosser, "Map-Reduce Meets Wider Varieties of Applications," *Intel Research Pittsburgh*, IRP-TR-08-05, 2008.
[17] K. S. Cho *et al.*, "Opinion Mining in MapReduce Framework," *Secure and Trust Computing, Data Management, and Applications*, 2011, pp. 50-55.
[18] R. Cordeiro *et al.*, "Clustering Very Large Multi-dimensional Datasets with MapReduce," in *Proc. the International Conference onKnowledge Discovery and Data Mining*, 2011, pp. 690-698.
[19] M. Niemenmaa, A. Kallio, A. Schumacher, E. Klemela, and K. Heljanko, "Hadoop-BAM: directly manipulating next generation sequencing data in the cloud," *Oxford Journals, Bioinformatics*, vol. 28,
no. 6, pp. 876-877, 2012.
[20] J. Chandar, "Join Algorithms using Map/Reduce," M.S. thesis, School of Informatics, University of Edinburgh, United Kingdom 2010.
[21] N. S. Srirama, P. Jakovits, and E. Vainikko, "Adapting scientific computing problems to clouds using MapReduce," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 184-192, 2012.
[22] L. Kaufman and P. Rousseeuw, *Finding Groups in Data An Introduction to Cluster Analysis*. New York: Wiley Interscience, 1990.
[23] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1994.
[24] D. Jiang, B. Ooi, L. Shi, and S. Wu, "The Performance of MapReduce: An In-depth Study," *PVLDB*, vol. 3, no. 1, pp. 472-483, 2010.
[25] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: Efficient Iterative Data Processing on Large Clusters," *PVLDB*, vol. 3, no. 1, pp. 285--296, 2010.
[26] E. Elnikety, T. Elsayed, and H. E. Ramadan," iHadoop: Asynchronous Iterations for MapReduce," in *Proc. the 2011 IEEE Third InternationalConference on Cloud Computing Technology and Science*, 2011, pp.81- 90.
[27] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "iMapReduce: A Distributed Computing Framework for Iterative Computation," in *Proc. the 2011 IEEE International Symposium on Parallel and Distributed Processing
Workshops and PhD Forum*, 2011, pp. 1112-1121.
[28] A. Dave, W. Lu, J. Jackson, and R. Barga, "CloudClustering: Toward an iterative data processing pattern on the cloud," in *Proc. the 2011 IEEE International Symposium on Parallel and Distributed Processing
Workshops and PhD Forum*, 2011, pp. 1132-1137.
[29] J. Ekanayake *et al.*, "Twister: A Runtime for Iterative MapReduce," in *Proc. the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 810-818.
[30] A. Thusoo *et al.*, "Hive – A Petabyte Scale Data Warehouse Using Hadoop," in *Proc. 26th IEEE International Conference on Data Engineering*, Long Beach, California, 2010, pp. 996-1005.