World Journal of Engineering Research and Technology
WJERT
www.wjert.org
SJIF Impact Factor: 3.419

OPTIMIZATION OF MAP REDUCE APPLICATIONS USING PARTITION AND AGGREGATION IN BIG DATA APPLICATION

Malatesh S.H, Jyothi Kanaka Durga Kasa, Mushtaq Ahmed, Rahul Reddy, Taniya Chadha*

CSE, MSEC, Bangalore, India.

Article Received on 10/04/2016 **Article Revised on 01/05/2016** **Article Accepted on 21/05/2016**

***Corresponding Author**
Taniya Chadha
CSE, MSEC, Bangalore, India.

ABSTRACT

Cloud computing, rapidly emerging as a new computation concept, offers agile and scalable resource access in a utility-like fashion, particularly for the processing of big data. An important open problem here is too effectively progress the data, from various geographical locations more time, into a cloud for efficient processing. Big Data introduces to datasets whose sizes are beyond the capability of typical database software tools to capture, accumulate, maintain and examined. The application of Big Data differs across verticals since of the several challenges that bring about the various use cases. With the increasing amount of data and the availability of high performance and relatively low-cost hardware, database systems have been extended and parallelized to run on multiple hardware platforms to manage scalability. Recently, a new distributed data processing framework called Map Reduce was proposed whose fundamental idea is to simplify the parallel processing using a distributed computing platform that offers only two interfaces. To further reduce network traffic within a Map Reduce job, we consider to aggregate data with the same keys before sending them to remote reduce tasks. Map Reduce is a framework for processing and managing large scale data sets in a distributed cluster, which has been used for applications such as generating search indexes, document clustering, access log analysis, and various other forms of data analytics. In existing system, a hash function is used to partition intermediate data among reduce tasks. In this project the system proposed a decomposition-based distributed algorithm to deal with the large-scale optimization problem for big data

application and an online algorithm is also designed to adjust data partition and aggregation in a dynamic manner.

KEYWORDS: Aggregation, Hadoop, Job Scheduling, Map Reduce, Partitioning.

1. INTRODUCTION

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. The core of Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part called Map Reduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed. This approach takes advantage of data locality nodes manipulating the data they have access to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking. Hadoop consists of the Hadoop Common package, which provides file system and OS level abstractions, a Map Reduce engine and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the necessary Java Archive (JAR) files and scripts needed to start Hadoop.
The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. A Hadoop cluster has nominally a single name node plus a cluster of data nodes, although redundancy options are available for the name node due to its criticality. Each data node serves up blocks of data over the network using a block protocol specific to HDFS. The file system uses TCP/IP sockets for communication. Clients use remote procedure call (RPC) to communicate between each other. Map Reduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.

Map Reduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function. A Map Reduce program is composed of a Map () procedure that performs filtering and sorting and a Reduce () method that performs a summary operation. The Map Reduce System orchestrates the processing by marshalling the distributed servers, running the

various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

Above the file systems comes the Map Reduce Engine, which consists of one Job Tracker, to which client applications submit Map Reduce jobs. The Job Tracker pushes work out to available Task Tracker nodes in the cluster, striving to keep the work as close to the data as possible. With a rack-aware file system, the Job Tracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network. If a Task Tracker fails or times out, that part of the job is rescheduled. The Task Tracker on each node spawns a separate Java Virtual Machine process to prevent the Task Tracker itself from failing if the running job crashes its JVM. A heartbeat is sent from the Task Tracker to the Job Tracker every few minutes to check its status. The Job Tracker and Task Tracker status and information is exposed by Jetty and can be viewed from a web browser. By default Hadoop uses FIFO scheduling, and optionally 5 scheduling priorities to schedule jobs from a work queue.

The Map Reduce programming model simplifies large-scale data processing on commodity cluster by exploiting parallel map tasks and reduce tasks. Although many efforts have been made to improve the performance of Map Reduce jobs, they ignore the network traffic generated in the shuffle phase, which plays a critical role in performance enhancement. Traditionally, a hash function is used to partition intermediate data among reduce tasks, which, however, is not traffic-efficient because network topology and data size associated with each key are not taken into consideration.

2. RELATED WORK

In existing system, the system propose a Map Reduce algorithm to solve the ER problem for a huge collection of entities with multiple keys. The algorithm is characterized in the combination-based blocking and the load-balanced matching. The combination-based blocking utilizes the multiple keys to sort out necessary entity pairs for future matching. The load-balanced matching evenly distributes the required similarity computations to all the reducers in the matching step so as to remove the bottleneck of skewed matching computations for a single node in a Map Reduce framework.

The effectiveness and scalability of Map Reduce-based implementations of complex data-intensive tasks depend on an even redistribution of data between maps and reduce tasks. In the presence of skewed data, sophisticated redistribution approaches thus become necessary to achieve load balancing among all reduce tasks to be executed in parallel. For the complex problem of entity resolution, we propose and evaluate two approaches for such skew handling and load balancing. The approaches support blocking techniques to reduce the search space of entity resolution, utilize a preprocessing Map Reduce job to analyze the data distribution, and distribute the entities of large blocks among multiple reduce tasks. The evaluation on a real cloud infrastructure shows the value and effectiveness of the proposed load balancing approaches. In this paper, we propose and evaluate two effective load balancing approaches to data skew handling for MR-based entity resolution.

Note that MR's inherent vulnerability to load imbalances due to data skew is relevant for all kind of pairwise similarity computation, e.g., document similarity computation and set-similarity joins. Such applications can therefore also benefit from our load balancing approaches though we study MR-based load balancing in the context of ER only. The actual execution of an MR program (also known as job) is realized by an MR framework implementation such as Hadoop. An MR cluster consists of a set of nodes that run a fixed number of map and reduce processes. For each MR job execution, the number of map tasks (m) and reduce tasks (r) is specified. Note that the partition function part relies on the number of reduce tasks since it assigns key-value pairs to the available reduce tasks. Each process can execute only one task at a time. After a task has finished, another task is automatically assigned to the released process using a framework-specific scheduling mechanism.

In Fig 2.1 load balancing is mainly realized within the map phase of the second MR Job. Both strategies follow the idea that map generates a carefully constructed composite key that (together with associated partition and group functions) allows a balanced load distribution. The composite key thereby com-bines information about the target reduce task(s), the block of the entity, and the entity itself. While the MR partitioning may only use part of the map output key for routing, it still groups together key-value pairs with the same blocking key component of the composite key and, thus, makes sure that only entities of the same block are compared within the reduce phase.

As we will see, the map function may generate multiple keys per entity if this entity is supposed to be processed by multiple reduce tasks for load balancing. Finally, the reduce

phase performs the actual ER and computes match similarities between entities of the same block. Since the reduce phase consumes the vast majority of the overall runtime (more than 95% in our experiments), our load balancing strategies solely focus on data redistribution for reduce tasks. Other MR-specific performance factors are therefore not considered. For example, consideration of data locality would have only limited impact and would require additional modification of the MR framework. We utilized two real-world datasets.

Note that the blocking attributes were not chosen to artificially generate data skew but rather reflect a reasonable way to group together similar entities. Two entities were compared by computing the edit distance of their title. Two entities with a minimal similarity of 0.8 were regarded as matches. We proposed two load balancing approaches, Block Split and Pair Range, for parallelizing blocking-based entity resolution using the widely available Map Reduce framework. Both approaches are capable to deal with skewed data (blocking key) distributions and effectively distribute the workload among all reduce tasks by splitting large blocks. Our evaluation in a real cloud environment using realworld data demonstrated that both approaches are robust against data skew and scale with the number of available nodes. The Block Split approach is conception-wise simpler than Pair Range but achieves already excellent results. Pair Range is less dependent on the initial partitioning of the input data and slightly more scalable for large match tasks.

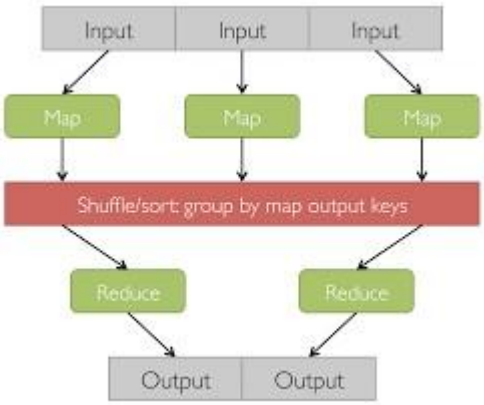

Fig 2.1. Illustrates the Map Reduce Algorithm of Hadoop system.

2.1. Disadvantages

- The existing works focuses on load balance at reduce tasks.
- It ignoring the network traffic during the shuffle phase.
- It can be only applied to the network topology with servers directly linked to other servers, which is of limited practical use.

- This approach is not traffic efficient.
- Hash based work load gives unstable work load.

3. PROPOSED SYSTEM

The system proposed a distributed algorithm for big data applications by decomposing the original large-scale problem into several sub problems that can be solved in parallel. The system design an online algorithm whose basic idea is to postpone the migration operation until the cumulative traffic cost exceeds a threshold. The network topology is based on three-tier architectures: an access tier, an aggregation tier and a core tier. The system investigate network traffic reduction within Map Reduce jobs by jointly exploiting traffic-aware intermediate data partition and data aggregation among multiple map tasks. It offers computers as physical or more often as virtual machines (VMs).

A cluster of VMs, virtual cluster, is often requested as a platform for users to run parallel or distributed applications such as Map Reduce and Dryad applications. In order to get high throughput, fast response, load balance, low cost, and low price, many topics on VM configuration, VM placement, VM consolidation, and VM migration are explored. The network topology of a virtual cluster has a significant impact on the execution of applications running on it because the physical nodes where VMs are located can be linked in different ways. For example, some nodes are located in the same rack while others in different racks through a slow link.

Another special architecture is the hierarchical network where two physical nodes may lie in different local area networks. Furthermore, the characteristics of different applications have different requirements for the network topology. Some applications create tasks running on different VMs which need to exchange large amount of data frequently, while others create tasks which execute independently or exchange a little data. Map Reduce and Map Reduce-like models are widely used to process “Big Data”. Applications based on such models place heavily datadependency or communication on VMs, so network traffic becomes the bottleneck of jobs. The following are three phases of data exchange in the execution process of an application based on Map Reduce model.

This paper targets at provisioning a virtual cluster according to the position relationship between VMs so as to decrease the network traffic and improve the performance of Map Reduce and Map Reduce-like applications rather than modifying the job scheduling strategies

or VM configurations. By optimizing the architecture of virtual clusters, cloud users can get a more efficient platform with the same resource request and cost, and cloud providers can obtain a higher resource utilization ratio. The shorter the distance, the closer the virtual cluster. The shortest distance problem is presented to obtain the closest virtual cluster.

We solve the shortest distance problem by formulating it into an integer linear programming. The online heuristic VM placement algorithm and the global sub-optimization algorithm are compared by simulations. The former has lower time complexity while the latter returns shorter average distance for multiple requests. We analyze the performance of our approach through experiments. In the experiment, we adopt different virtual cluster architectures to test different Map Reduce applications. Two metrics of application runtime and cluster affinity show the efficiency of virtual cluster optimization. The distance difference between two central node selection strategies.

Heuristic distance is mapped to the virtual cluster with the most appropriate central node built by our online VM placement algorithm. Shortest distance with a random central node is mapped to the same virtual cluster, however, the central node is chosen randomly. It is clear that even if we select the same virtual cluster, but not the same position of the central node, the distance difference is also great. For Map Reduce applications, the selection of the central node is the same important with the architecture of the virtual cluster. Each physical node has different capability and each request has different requirement.

For Map Reduce applications, it is very important to match the request with a virtual cluster and appropriate central node so as to make full use of the advantage of data-locality and reduce the network traffic. Fig. 2.1 shows the different distances under the different central nodes. We can see that the choice of the central node has an important impact on the distance for one request. This is because Map Reduce and Map Reduce-like models are based on master-slaves network topology. When the requests arrive randomly, their service time are also random, and the cloud resources are enough to meet multiple requests at one time, we can get the virtual cluster provisioning according to our online heuristic algorithm and global sub-optimization algorithm.

3.1 Advantages

- This algorithm is traffic efficient.
- Data partition and aggregation performed in dynamic manner.

- Network traffic cost is reduced in this approach.
- This is the first to propose the scheme that exploits both aggregator placement and traffic-aware partitioning.
- Each reduce task aggregates the related data partitions belonging to it and stores its result in the distributed file system.

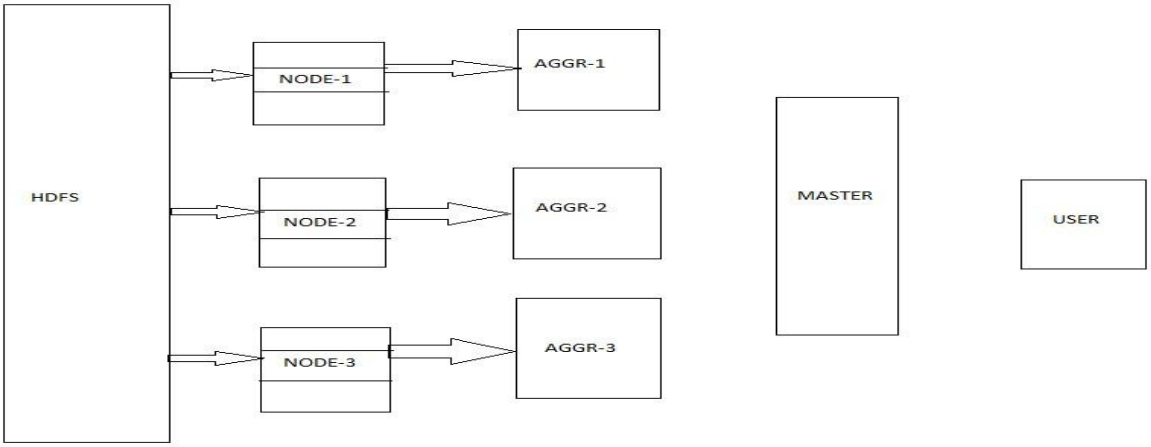

Fig 3.1. Proposed System Architecture

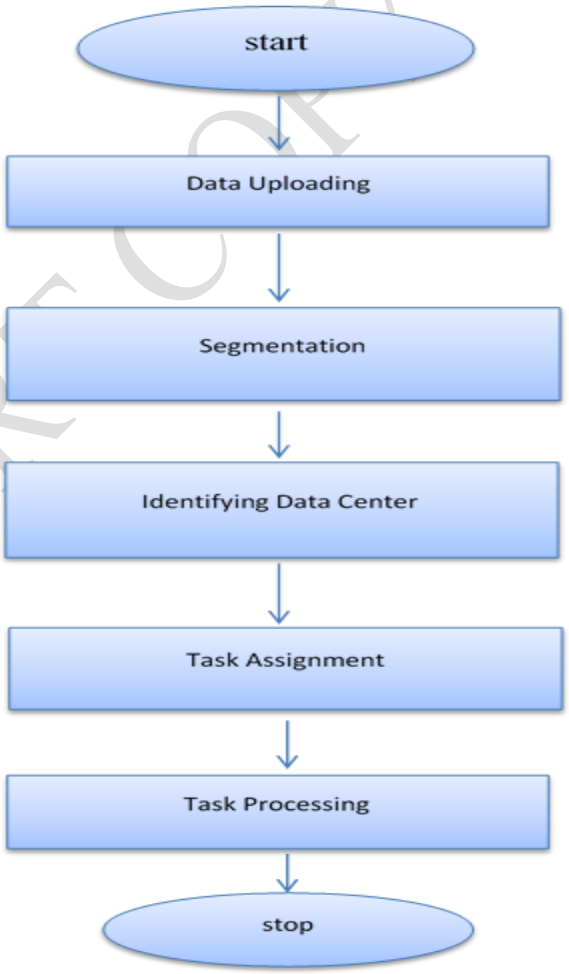

Fig 3.2. Data Flow Diagram for Proposed System

4. RESULT AND ANALYSIS

Through this project we plan to solve the below two problems:

- Network traffic: by designing a novel intermediate data partition scheme we aim to reduce network traffic cost.
- Aggregator placement problem: A decomposition based distributed algorithm is proposed to with the largescale optimization problem for big data application.

5. CONCLUSION

Map Reduce job scheduling. It is a hot topic to improve the performance of Map Reduce applications. Traditional and efficient locality-aware scheduling strategies can reduce processing time and increase throughput by decreasing the data traffic between nodes and balancing the loads of the cluster. The shuffle operation of Map Reduce model is proved to be as much as a third of the completion time of a Map Reduce job with others are map operation and reduce operation. The proposed system also presents the effectiveness of the relationship of task precedence and dependence.

6. ACKNOWLEDGEMENT

We express deepest gratitude to our project guide Prof. MALATESH S.H, Head of the Department, Computer Science and Engineering, M S Engineering College. We would like to thank our Principal Dr. K.S BADRINARAYANA who provide a healthy environment for all of us to work in best possible way. We also express our deep gratitude towards all the people who have helped us to completion of this project successfully.

7. REFERENCES

1. J. Dean and S. Ghemawat, “Map reduce: simplified data processing on large clusters,” Communications of the ACM, 2008; 51(1): 107–113.
2. W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, “Map task scheduling in map reduce with data locality: Throughput and heavy-traffic optimality,” in INFOCOM, 2013 Proceedings IEEE. IEEE, 2013, pp. 1609–1617.
3. F. Chen, M. Kodialam, and T. Lakshman, “Joint scheduling of processing and shuffle phases in map reduce systems,” in INFOCOM, 2012 Proceedings IEEE. IEEE, 2012, pp. 1143–1151.
4. Y. Wang, W. Wang, C. Ma, and D. Meng, “Zput: A speedy data uploading approach for the hadoop distributed file system,” in Cluster Computing (CLUSTER), 2013 IEEE International Conference on. IEEE, 2013, pp. 1–5.

5. T. White, Hadoop: the definitive guide: the definitive guide. ” O’Reilly Media, Inc.”, 2009.
6. S. Chen and S. W. Schlosser, “Map-reduce meets wider varieties of applications,” Intel Research Pittsburgh, Tech. Rep. IRP-TR-08-05, 2008.
7. J. Rosen, N. Polyzotis, V. Borkar, Y. Bu, M. J. Carey, M. Weimer, T. Condie, and R. Ramakrishnan, “Iterative mapreduce for large scale machine learning,” arXiv preprint arXiv:1303.3517, 2013.
8. S. Venkataraman, E. Bodzsar, I. Roy, A. AuYoung, and R. S. Schreiber, “Presto: distributed machine learning and graph processing with sparse matrices,” in Proceedings of the 8th ACM European Conference on Computer Systems. ACM, 2013, pp. 197–210.
9. A. Matsunaga, M. Tsugawa, and J. Fortes, “Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications,” in eScience, 2008. eScience’08. IEEE Fourth International Conference on. IEEE, 2008, pp. 222–229.
10. J. Wang, D. Crawl, I. Altintas, K. Tzoumas, and V. Markl, “Comparison of distributed data-parallelization patterns for big data analysis: A bioinformatics case study,” in Proceedings of the Fourth International Workshop on Data Intensive Computing in the Clouds (Data Cloud), 2013.
11. R. Liao, Y. Zhang, J. Guan, and S. Zhou, “Cloudnmf: A mapreduce implementation of nonnegative matrix factorization for largescale biological datasets,” Genomics, proteomics & bioinformatics, 2014; 12(1): 48–51.
12. G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, “Introducing map-reduce to high end computing,” in Petascale Data Storage Workshop, 2008. PDSW’08. 3rd. IEEE, 2008, pp. 1–6.
13. W. Yu, G. Xu, Z. Chen, and P. Moulema, “A cloud computing based architecture for cyber security situation awareness,” in Communications and Network Security (CNS), 2013 IEEE Conference on. IEEE, 2013, pp. 488–492.
14. J. Zhang, H. Zhou, R. Chen, X. Fan, Z. Guo, H. Lin, J. Y. Li, W. Lin, J. Zhou, and L. Zhou, “Optimizing data shuffling in dataparallel computation by understanding user-defined functions,” in Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI), San Jose, CA, USA, 2012.
15. F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, “Mapreduce with communication overlap,” 2013; pp. 608–620.

16. H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Mapreduce-merge: simplified relational data processing on large clusters,"in Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, 2007, pp. 1029–1040.
17. T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth,J. Talbot, K. Elmeleegy, and R. Sears, "Online aggregation and continuous query support in mapreduce," in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010, pp. 1115–1118.
18. A. Blanca and S. W. Shin, "Optimizing network usage in mapreduce scheduling."
19. B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: localityaware resource allocation for mapreduce in a cloud," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011, p. 58.
20. S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in Cloud Computing Technology and Science (Cloud Com), 2010 IEEE Second International Conference on. IEEE, 2010, pp. 17–24.
21. L. Fan, B. Gao, X. Sun, F. Zhang, and Z. Liu, "Improving the load balance of mapreduce operations based on the key distribution of pairs," arXiv preprint arXiv: 1401.0355, 2014.
22. S.-C. Hsueh, M.-Y. Lin, and Y.-C. Chiu, "A load-balanced mapreduce algorithm for blocking-based entityresolution with multiple keys," Parallel and Distributed Computing 2014, p. 3.
23. T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online." in NSDI, vol. 10, no. 4, 2010, p. 20.
24. J. Lin and C. Dyer, "Data-intensive text processing with mapreduce," Synthesis Lectures on Human Language Technologies, 2010; 3(1); 1–177.
25. P. Costa, A. Donnelly, A. I. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications." In NSDI, 2012; 12: 3–3.
26. Huan Ke, Peng Li, Song Guo, and Minyi Guo, "On Traffic-Aware Partition and Aggregation in MapReduce for Big Data Applications".
27. Apache Hadoop - Wikipedia, the free encyclopedia https://en.wikipedia.org/wiki/Apache_Hadoop.
28. hadoop.apache.org/