Traffic-Aware Partition and Aggregation in Map Reduce for Big Data Applications

M. Supriya
M.E Student,
Department of Computer Science and Engineering,
Institute of Road and Transport Technology,
Erode.

***Abstract*:- The purpose of the system to cut down network traffic cost. Map out a novel intermediate data participant schema. The map reduce type simplifies the large scale data deal with product group even though many times effort have been made to maximize the execution of map reduce work. They ignore the network effects which conclude. A fundamental part in implementation update. A hash capacity made use of session middle of the topology among minimizes the task even so is note movement valued in network topology. At last board reproduced outcome that shows at the proposed algorithm can together minimize network cost under both offline and online cases.**

Keywords*— *Data Aggregation, Decomposition-based distributed Algorithm*, *Big Data, Online Hash-based Partition with Aggregation, Online Hash-based Partition with Random Aggregation.

I INTRODUCTION

Map Reduce is one of the most popular computer frameworks for big data processing. Hadoop is a java based programming language. Hadoop having map reduce and hadoop distributed file system. Map Reduce divides a computation into map and reduce, which carried out by several map tasks and reduce tasks, respectively. In the map phase, map tasks are launched in parallel to convert the original input splits into intermediate data in a form of key/value pairs. Local machine stored all the key/value pair and organized into multiple data partitions. Each cut off task fetches its own share of data partitions from all map tasks to create the final result. There will be a shuffle step between map and reduce phase. The data produced by the map phase are ordered cause of data and transferred to the appropriate machines implement the reduce phase. The outcome of network traffic pattern are all map tasks to all reduce tasks can create a great volume of network traffic, imposing a serious control the efficiency of data analytic Applications. By default, intermediate data are shuffled according to a hash function in Hadoop. This would lead to vast network traffic because it ignores network topology and data size associated with each key. It manipulates right away after a map task solely for its cause data, fails to use the data aggregation opportunities among multiple tasks on different machines. Map Reduce job purpose is to minimize the total network traffic.

II RELATEDWORK

Related work on mapreduce performance improvement by optimizing data by lead of an optimizing network usage. System performance better and found but high network utilization and network congestion occurred simultaneously with good performance. Network traffic can be reduced by locality-awareness in the shuffle phase generated in the cloud data center.
Distribution of intermediate key/value pairs to improve the load balance for ignoring the network traffic during the shuffle phase. Mapreduce job have introduced a combine function but reduces the amount of data to due shuffled and merged reduce task.

III EXISTING WORK

In existing system, the system proposes a Map Reduce algorithm to solve the ER problem for a enormous 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 eliminate 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 a just redistribution of data between maps and reduce tasks.

IV PROPOSED WORK

Map Reduce is based programming model with map function and reduce function. This Map Reduce job is ruined over a distributed system composed at a master and a set of workers input. It is divided into chunks that are assigned to map tasks. Output divided into as many partitions as the number of reducers for the job.
The network traffic of the DDA algorithm is about 3:4105, while the traffic cost of our algorithm is only 1:7 105, with a decrement of 50%. In contrast to DDA and DDA, the curve of DA amplify slowly because most map outputs are aggregated and traffic-aware partition chooses closer reduce tasks for each key/value pair, which are beneficial to network reduction in the shuffle phase.

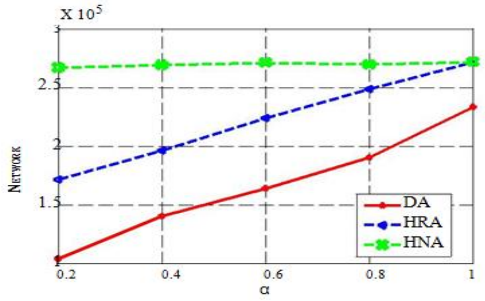

Fig.1. Network traffic cost versus data reduction ratio

The performance of three algorithms under different values of changing its value from 0.2 to 1.0. A small value of indicates a lower aggregation efficiency for the intermediate data. We observe that network traffic increases as the growth of under both DA and HRA. When is 0.2, DA carry out the lowest traffic cost of 1:1 105. On the other hand, network traffic of HNA keeps stable because it does not conduct data aggregation.

V DISTRIBUTED ALGORITHM DESIGN

Distributed algorithm design using highly efficient approximation algorithms is solving an additional challenge in big data for map reduce job. An additional challenge arises in dealing with the map Reduce job for big data. The problem on multiple machines in a parallel manner is solved using distributed algorithm design.

A. Network Traffic Traces

Distributed algorithm can be using real trace in a cluster of 5 virtual machines with 1GB memory and 2GHz CPU.

This network topology is based on three tier architectures: an access tier, an aggregation tier and a core tier. The access tier is made up of cost effective Ethernet switches connecting rack VMs. The access switches are connected via Ethernet to a set of aggregation switches which in turn are connected to a layer of core switches. An inter rack link is the most contentious resource as all the VMs hosted on a rack transfer data across the link to the VMs on other racks.

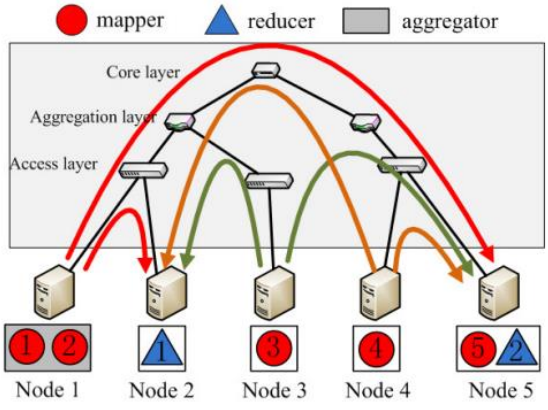

Fig.2. three tier architecture

Real data source from latest dumps files in Wikimedia are used to test real network traffic cost in hadoop.

VI ONLINE ALGORITHM

In online algorithm before starting reduce tasks. We need to wait all mapper to finish their process.

OPT ONE SHOT:

Min CM(t) + X p∈P C p (t) Map and reduce tasks may partially overlap in execution to maximize system throughput; it is hard to assess system parameters at a high accuracy for big data applications.

VII PERFORMANCE COSIDERATION

To evaluate the performance of our proposed distributed algorithm DA. We compare DA with HNA, and compared with the HRA.

- **HNA:** Hash-based partition with No Aggregation. It exploits the traditional hash partitioning for the intermediate data. It is the default method in Hadoop.
- **HRA:** Hash-based partition with Random Aggregation. It adds a random aggregator placement algorithm based on the traditional Hadoop. It aims to reducing the network traffic cost in the comparison of traditional method in Hadoop.

A. Simulation results of offline cases

Distributed algorithm and the optimal solution obtained by solving the MILP formulation. Due to the high computational complexity of the MILP formulation. Performance of our distributed algorithm is very close to the optimal solution. Although network traffic cost increases as the number of keys grows for all algorithms, the performance enhancement of our proposed algorithms to the other two schemes becomes larger contrast to HRA and HNA, the curve of DA increases slowly because more map outputs are aggregated and traffic-aware partition chooses closer reduce tasks for each key/value pair reduce the network traffic reduction in the shuffle phase.

B. Process outcome of online cases

We noticed that network traffic rises at first and tend.

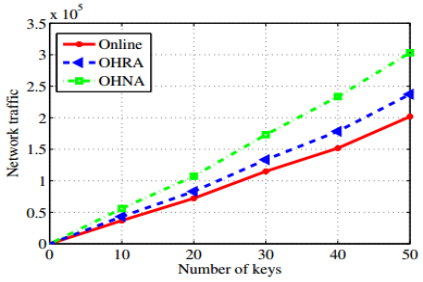

Fig.3. network traffic cost versus number of keys

To became stable according to our online algorithm. Network traffic of OHRA and OHNA keep stable constantly because OHNA follow the same hash partition scheme. We notice that online algorithm work much better than two algorithms.

VIII CONCLUSIONS

The joint optimization of intermediate data partition and aggregation in Map Reduce to minimize network traffic cost for big data applications. Map Reduce to decrease network traffic cost for big data applications. We programming distributed algorithm to clear a problem on multiple machine in additional. We extended our algorithm to deal with map reduce job in a online manner. The simulation results demonstrate that our proposals can effectively reduce network traffic cost under various network settings.

IX FUTURE ENHANCEMENT

Fault tolerance mechanisms consume major extra energy to detect and recover from the failures. Fault tolerant describes a computer system or component designed. so that, component fails, in that event a backup component or procedure can immediately take place with no loss of service. Fault tolerance is not just a function of individual machines; it may also characterize the rules by which they interact. Fault tolerance is the function that enables a system to proceeding operating properly in the event of the failure of some of its components.

REFERENCES

[1] J. Dean and S. Ghemawat, "Map reduce: simplified data processing on large clusters," Communications of the ACM, pp. 107–113, 2008.
[2] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Map reduce online." 2010.
[3] T. White, Hadoop: the definitive guide: the definitive guide. "O'Reilly Media, Inc.", 2009.
[4] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijay Kumar, "Map reduce with communication overlap," pp. 608–620, 2013.
[5] F. Chen, M.Kodialam, and T. Lakshman, "Joint scheduling of processing and shuffle phases in mapreduce systems," in INFOCOM, 2012 Proceedings IEEE. IEEE, 2012, pp. 1143–1151.
[6] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in INFOCO, IEEE, 2013, pp. 1609–1617.
[7] R. Liao, Y. Zhang, J. Guan, and S. Zhou, "Cloudnmf: A mapreduce implementation of nonnegative matrix factorization for large-scale biological datasets," Genomics, proteomics & bioinformatics, vol. 12, no. 1, pp. 48–51, 2014.
[8] S.C. Hsueh, M.-Y. Lin, and Y.-C. Chiu, "A load-balanced map reduce algorithm for blocking-based entity resolution with multiple keys, "Parallel and Distributed Computing 2014, p. 3, 2014.
[9] J. Lin and C. Dyer, "Data-intensive text processing with map reduce, "Synthesis Lectures on Human Language Technologies, vol. 3, no. 1, pp. 1–177, 2010.
[10] P. Costa, A. Donnelly, A. I. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications." In NSDI, vol. 12, 2012, pp. 3–3.
[11] 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 (CloudCom), IEEE, 2010, pp. 17– 24.
[12] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for mapreduce in a cloud," Networking, Storage and Analysis. ACM, 2011, p. 58.
[13] 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.
[14] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, "Introducing mapreduce to high end computing," PDSW'08. 3rd. IEEE, 2008, pp. 1–6.
[15] 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.
[16] S. Venkataraman, E. Bodzsar, I. Roy, A. AuYoung, and R. S. Schreiber, "Presto: distributed machine learning and graph processing with sparse matrices," in ACM, 2013, pp. 197– 210.
[17] A. Matsunaga, M. Tsugawa, and J. Fortes, "Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications," in eScience, 2008. IEEE, 2008, pp. 222–229.