International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue-01
January 2018

A Traffic Minimization Approach for Big Data in Map Reduce Job by Intermediate Data Partition Technique

K.Mounika & J.Rajashekar
mounikabtech.komirelli@gmail.com, Mail Id: rajcse504@gmail.com
[1] PG Scholar, Dept of CSE, VBIT College of engineering, Aushapur (v), Ghatkasar (m), Medchal Dist, Telangana, India,
[2] Assistant Professor, Dept of CSE, VBIT College of engineering, Aushapur (v), Ghatkasar (m), Medchal Dist, Telangana, India

ABSTRACT:
MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Scheduling map tasks to improve data locality is crucial to the performance of MapReduce. Many works have been devoted to increasing data locality for better efficiency. However, to the best of our knowledge, fundamental limits of MapReduce computing clusters with data locality, including the capacity region and theoretical bounds on the delay performance, have not been studied. we propose the on traffic aware partition and aggregation in order to reduce the network cost for map reduce jobs by designing an intermediate data partition scheme. Moreover, we together consider the aggregator placement issue, where each aggregator can reduce merged traffic from more than one map duties. A decomposition-primarily based distributed algorithm is proposed to address the large-scale optimization trouble for a big data application and an online algorithmic rule is also designed to adjust network data partition and aggregation in a dynamic way.

1. INTRODUCTION

Processing large-scale datasets has become an increasingly important and challenging problem as the amount of data created by online social networks, healthcare industry, scientific research, etc., explodes. MapReduce/Hadoop is a simple yet powerful framework for processing large-scale datasets in a distributed and parallel fashion, and has been widely used in practice, including Google, Yahoo!, Facebook, Amazon and IBM. A production MapReduce cluster may even consist of tens of thousands of machines .The stored data are typically organized on distributed file systems (e.g., Google File System (GFS) , Hadoop Distributed File System (HDFS), which divide a large dataset into data chunks (e.g., 64 MB) and store multiple replicas (by default 3) of each chunk on different machines. A data processing request under the MapReduce framework, called a job, consists of two types of tasks: map and reduce.

Map Task Execution:

Each map task is assigned a portion of the input file called a split. By default, a split contains a single HDFS block (64MB by default), so the size of the input file determines the number of map tasks. The execution of a map task is divided into two phases. The map phase reads the task's split from HDFS, parses it into records (key/value pairs), and applies the map function to each record. After the map function has been applied to each input record, the commit phase registers the final output with the TaskTracker, which then informs the JobTracker that the task has finished executing. After a map task has applied the map function to each input record, it enters the commit phase. To generate the task's final output, an in-memory buffer is flushed to disk, and

all of the spill files generated during the map phase are merge sorted into a single data file. The final output file is registered with the TaskTracker before the task completes. The TaskTracker will read these files when servicing requests from reduce tasks.

Reduce Task Execution:

The execution of a reduce task is divided into three phases. The shuffle phase fetches the reduce task's input data. Each reduce task is assigned a partition of the key range produced by the map step, so the reduce task must fetch the content of this partition from every map task's output. The sort phase groups records with the same key together. The reduce phase applies the user-defined reduce function to each key and corresponding list of values. In the shuffle phase, a reduce task fetches data from each map task by issuing HTTP requests to a configurable number of TaskTrackers at once (5 by default). The JobTracker relays the location of every TaskTracker that hosts map output to every TaskTracker that is executing a reduce task. In traditional batch-oriented Hadoop, a reduce task cannot fetch the output of a map task until the map has finished executing and committed its final output to disk. After receiving its partition from all map outputs, the reduce task enters the sort phase. The map output for each partition is already sorted by key. The reduce task merges these runs together to produce a single run that is sorted by the key. The task then enters the reduce phase, in which it invokes the user-defined reduce function for each distinct key in sorted order, passing it the associated list of values. The output of the reduce function is written to a temporary location on HDFS. After the reduce function has been applied to each key in the reduce task's partition, the task's HDFS output file is atomically renamed from its temporary location to its final location.

In this paper, we jointly consider data partition and aggregation for a MapReduce job with an objective that is to minimize the total network traffic. In particular, we propose a distributed algorithm for big data applications by decomposing the original large-scale problem into several subproblems that can be solved in parallel. Moreover, an online algorithm is designed to deal with the data partition and aggregation in a dynamic manner. Finally, extensive simulation results demonstrate that our proposals can significantly reduce network traffic cost in both offline and online cases.

2.RELATED WORK

J. Dean and S. Ghemawat explained MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. Our

implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

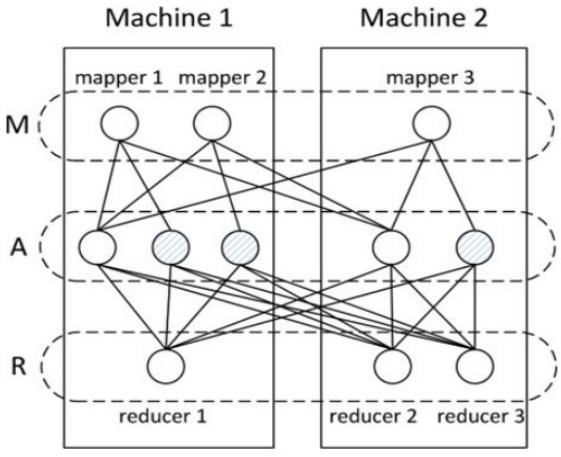
W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, explained Scheduling map tasks to improve data locality is crucial to the performance of MapReduce. Many works have been devoted to increasing data locality for better efficiency. However, to the best of our knowledge, fundamental limits of MapReduce computing clusters with data locality, including the capacity region and theoretical bounds on the delay performance, have not been studied. In this paper, we address these problems from a stochastic network perspective. Our focus is to strike the right balance between data-locality and load-balancing to simultaneously maximize throughput and minimize delay. We present a new queuing architecture and propose a map task scheduling algorithm constituted by the Join the Shortest Queue policy together with the MaxWeight policy. We identify an outer bound on the capacity region, and then prove that the proposed algorithm stabilizes any arrival rate vector strictly within this outer bound. It shows that the algorithm is throughput optimal and the outer bound coincides with the actual capacity
Region. Further, we study the number of backlogged tasks under the proposed algorithm, which is directly related to the delay performance based on Little's law. We prove that the proposed algorithm is heavy-traffic optimal, i.e., it asymptotically minimizes the number of backlogged tasks as the arrival rate vector approaches the boundary of the capacity region. Therefore, the proposed algorithm is also delay optimal in the heavy-traffic regime.

F. Chen, M. Kodialam, and T. Lakshman proposed MapReduce has achieved tremendous success for large-scale data processing in data centers. A key feature distinguishing MapReduce from previous parallel models is that it interleaves parallel and sequential computation. Past schemes, and especially their theoretical bounds, on general parallel models are therefore, unlikely to be applied to MapReduce directly. There are many recent studies on MapReduce job and task scheduling. These studies assume that the servers are assigned in advance. in current data centers, multiple MapReduce jobs of different Importance levels run together. In this paper, we investigate a schedule problem for MapReduce taking server assignment in to consideration as well. We formulate a MapReduce server-job organizer problem (MSJO) and show that it is NP-complete. We develop a 3-approximation algorithm and a fast heuristic. We evaluate our algorithms through both simulations and experiments on Amazon EC2 with an implementation in Hadoop. The results confirm the advantage of our algorithms

FRAMEWORK

The network traffic is reducing the inside of a MapReducejob. Wehave to consider the aggregate information with similar keys since mailing them to remote reduce tasks. Although we've asimilar perform, referred to as combiner, that has been

alreadyadopted by Hadoop, it operates directly when a map taskindividually for its generated information, failing to use the informationaggregation opportunities among multiple tasks on completely differentmachines.Objective is to reduce the overall network traffic byData partition and aggregation for a MapReduce job as shownin fig. Distributed algorithmic program is planned for big dataapplications by decomposing the initial large-scale drawbackinto many subproblems and these subproblems is solvedin parallel. Another is on-line algorithmic program that is additionally designed to deal with the information partition and aggregation during a dynamicmanner.Finally demonstrated suggest thatour proposals will considerably reduce network traffic price inboth offline and on-line cases. The network trafficminimization drawback is being developed. To encourage ouranalysis, we want to construct an auxiliary graph with a layer structure.



Another extra challenge that arises in managingthe MapReduce job is for big data. To solve the problem onmultiple machines during a parallel manner we use distributedalgorithm. The only basic plan behind this can be to decomposethe original large-scale drawback into many distributedsolvable subproblems that are coordinated by a high-levelmaster drawback.

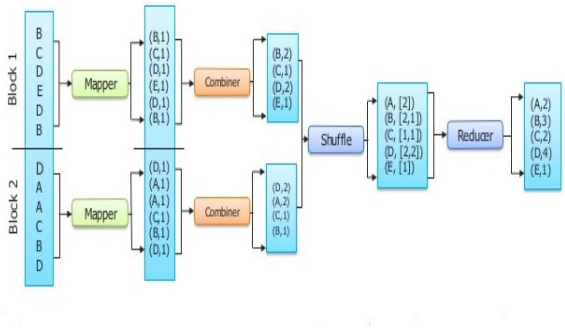


Fig.2. Map Reduce Task with Aggregation

Map and reduce tasks might partly overlap under somecases however the execution is to extend system throughput, and itis troublesome to estimate system parametersat a high accuracy forbig informationapplications.

A. Distributed Algorithm(rewrite this content)

We propose a distributed algorithm for big data applications by decomposing the original large-scale problem into several subproblems that can be solved in parallel.Our basic idea is to decompose the original large-scale problem into several distributively solvable sub problems that are coordinated by a high-level master problem.We executed our distributed algorithm using the same data source for comparison. Since our distributed algorithm is based on a known aggregation ratio a, we have done some experiments to evaluate it in Hadoop environment.

B. Online Algorithm(rewrite this content)

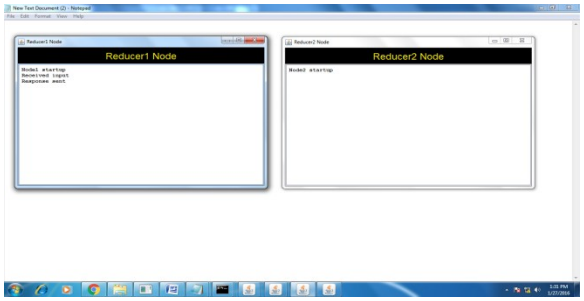
In this section, we design an online algorithm whose basic idea is to postpone the migration operation until

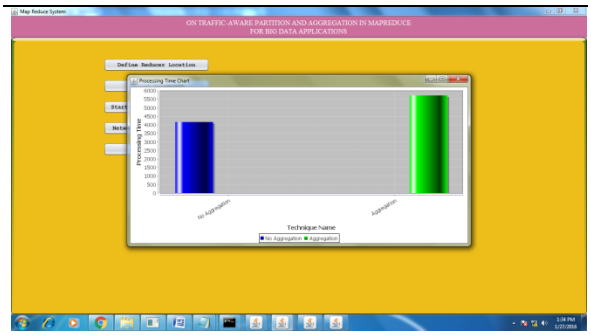
the cumulative traffic cost exceeds a threshold.In each of the following time slot, we check whether the accumulative traffic costis greater than g times of $C_{M(t\hat{})}$ .If it is, we solve an optimization problem with the objective of minimizing traffic cost.We conduct migration operation according to the optimization results and update $C_{M(t\hat{})}$.

A web algorithmic program to dynamically adjust information partition and aggregation throughout the execution ofmap and reduce tasks is therefore driven. The fundamental plan of thisalgorithm is to postpone the migration operation till thecumulative traffic price exceeds a threshold.Extensive simulations are carried toevaluate theperformance of our projected distributed algorithmic program Distributed Algorithm. We have a tendency to then compare DA with HNA that is that thedefault technique in Hadoop. To our best data, wepropose the placement algorithmic program, and comparedwith the HRA that focuses on a random somebody placement.

3. EXPERIMENTAL RESULTS

Network traffic by planning a novel intermediate information partition scheme we aim toreduce network traffic price.Aggregator placement is decomposition based mostly distributed algorithmic rule is plannedto with the largescale optimization problem for big data application.In reducer location you are givenfor particular location we are taking the latitude and longitude values,after upload any documents and start MapReduce Aggregation. The request has been processed by Reducers, because which reducer is nearer to the mapped location it receives input and response sent that reducer.Compare the aggregation and non aggregation network traffic cost graph.





4. CONCLUSION

The MapReduce programming model has been successfully used at Google for many different purposes. We attribute this success to several reasons. First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault tolerance, locality optimization, and load balancing.Those key/value pairs are saved on local system and organized into more than one data partitions, one per reduce challenge. Within the reduce section, every reduce task fetches its very own proportion of records partitions from all map tasks to generate the final end result. There may be a shuffle step between map and reduce segment. On this step, the information produced with the aid of the map phase are ordered, partitioned and transferred to the

suitable machines executing the reduce phase. The resulting network traffic patterns from all map responsibilities to all reduce tasks can motive a top notch volume of network visitors, imposing a critical constraint at the efficiency of data analytic applications and programs.To address big-scale formulation due to large data, we recommend a distributed algorithm to resolve the trouble on more than one machine. Moreover, we enlarge our algorithm to handle the MapReduce process in an online way while some system parameters are not given. Finally, we conduct significant simulations to evaluate our proposed set of rules beneath both offline instances and on-line cases.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters,"*Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] 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 *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 mapreduce 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 Conferenceon*. 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 ACMEuropean 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. IEEEFourth International Conference on*. IEEE, 2008, pp. 222–229.

[10] J. Wang, D. Crawl, I. Altintas, K. Tzoumas, andV. Markl, "Comparison of distributed data parallelization patterns for big data analysis: A bioinformatics case study," in*Proceedings of the FourthInternational Workshop on Data Intensive Computing in the Clouds (DataCloud)*, 2013.

[11] R. Liao, Y. Zhang, J. Guan, and S. Zhou, "Cloudnmf: A mapreduceimplementation of nonnegative matrix factorization for largescale biological datasets," *Genomics, proteomics &bioinformatics*, vol. 12, no. 1, pp. 48–51, 2014.

[12] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, "Introducing map-reduce to high end computing," in *PetascaleData 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 Conferenceon*. IEEE, 2013, pp. 488–492.