ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

Implementing Efficient Topology Aware System of Network Levitated Merge for Hadoop Acceleration

Madhura Sadashiv Dhekane, Prof. Gaikwad V. S.

Dept. of Computer Engineering, JSPM's Rajarshi Shahu School of Engineering and Research, JSPM Narhe Technical Campus, Pune, India

Dept. of Computer Engineering, JSPM's Rajarshi Shahu School of Engineering and Research, JSPM Narhe Technical Campus, Pune, India

ABSTRACT: Distributed systems are increasingly used these days. These systems consider several other machines working for serving a purpose as a single system. Such a system is generally designed for studying the concept of MapReduce and enhances the existing system by adding some functionality. Many researches are carried out on MapReduce as it's an effective implementation with hadoop is the most welcomed technology these days. A unusual kind of issues are noticed which can hamper the performance of the system. One of them is poor network maintain which may include wrong connections, low bandwidth etc. Or huge number of middle merges where more storage is accessed for many times. It is also applicable in the many circumstances where we have to address same data all over again. Avoiding these all kinds of situations the latency of the network can be reduced. Hopefully, this system will help out network levitated systems which use hadoop for handling the big data.

KEYWORDS: Hadoop, MapReduce, network-levitated merge, Hadoop acceleration, pipelining

I. INTRODUCTION

MapReduce is an implementation for handling and creating lagre data sets. This is more widely used in different applications of distributed systems, where large amount of data is handled. Distributed system is used in extraordinary ways where the functioning, services, databases are divided in number of parts and those are taken care of different requests by the users.

MapReduce can be used where big data can be generated by real time transactions or it may be stored using large data storage systems. The size of data may differ from Gigabytes of data to zeta bytes of data. The examples of companies which can be considered those continue to generate large amount of data like facebook~6million messages per day; ~ 2million page views a day by ebay, ~9petabytes of storage based satellite images by skybox imaging ~terabytes a day.

MapReduce programs are composed of two main procedures as Map() filtering and sorting of data items provided to it, and Reduce() which will achieve the summary operation. The fault tolerance ,scalability are the main benefits of MapReduce technic to be used in distributed systems effectively. MapReduce libraries are written as in some other levels of optimizations. This work is done in a few of the programming languages. So the portability and scalability can be achieved. MapReduce is used in increasing number of application domains, as it is more capable of handling large amounts of data and using less disk-oriented mechanisms that can give the results with good remarks.

A number of researches have been carried out to improve performance of MapReduce framework. Although the relationship in between phases of MapReduce are critical issue to be solved, MapReduce Online architecture developed by Condie et al.[3] comes with direct network channels between Map and Reduce operations. Several serialization issues are observed by different researchers. This is because the data elements are stored on different storage locations in the systems and also the intermediate results are also stored and accessed to and from the disks. These repetitive operations make the operation more time consuming and costly. To reduce this, Hadoop-A algorithm

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

is developed by Weikuan yu et al. They have implemented a full pipelined architecture for overlapping MapReduce operations, and have worked on reducing disk accesses. But there is a possibility of minimizing the memory accesses and also to develop a topology aware system that would minimize the latency in responding to the user.

The motivation to work on this topic is there are different issues in using the MapReduce as it is. Those issues are

1) The serialization between Hadoop shuffle/ merge and reduce phases.
2) Lack of network portability.
3) Repetitive disk accesses and data merges.

The system introduced in this paper works on minimizing the latency of operations, communications and network latency as being a topology aware system. It is more efficient to users as it gives the results in minimum times by storing data once in a lifetime of request-response cycle.

The organization of this paper is as follows, section II contains related work. This gives the work done previously by different researchers in this area. Section III describes system design of proposed system. It detailed implementation and working of the system. It is followed by results of the system in section IV, where all the graphs generated by the system are discussed. Finally, the topic is concluded in section V.

II. RELATED WORK

Weikuan yu, Yandong Wang, and Xinyu Que introduced a system with network levitated merge algorithm called Hadoop-A. This system works on network levitated merge and it uses pipelining for the three MapReduce phases those are shuffle, merge and reduce. But this system doesnot uses the hierarchical merging for memory scalability and they have network portability as their future scope[1].

Condie et al[3] have proposed an architecture called MapReduce Online to open up direct network channels between Map() and Reduce() and speed up the delivery of data between Map() to Reduce(). But look at the relationship of hadoop MapReduce's three data processing phases and their implication to the efficiency of Hadoop remains a critical issue.

Phoenix[7] is a MapReduce implementation for shared memory systems. In this system, the users have require to write simple parallel code without considering the complexity of thread creation, dynamic task, scheduling data partitioning[1], and fault tolerance across processor nodes[20].

Tiled MapReduce[9] by chen et al. improves Phoenix[7]. It leverages the tiling strategy that is commonly used in compiler community. The division of MapReduce is done into discrete subjobs and extends the Reduce() phase to process partial Map result. However enhancement of large scale hadoop clusters, and reduction of disk contention as an optimization strategy is not added to the system and optimization of inter-job pipelining is in stark.

III. SYSTEM DESIGN

The system is designed with three modules; those can be taken as a good example of distributed system. Here, the three modules can run on different machines. Also, we can have multiple number of user machine and multiple server agents working or different machines.

A. *Mathematical Model:*

a. Problem Description*:*

Let P be the network levitated merge system, which provides MapReduce in proposed form; such that

$P=\{U, S, A, Rq, Rs| \phi p\}$

Where,

- U represents number of system users; $U= \{U_0,U_1,U_2,\ldots,U_n| \phi u \}$.
- U is designed such that for each $U_i=\{UI, UV, Op, DI| \phi u\}$.

Where,

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

- UI is uploaded image.
- UV is uploaded video.
- Op represents operation to be performed.
- DI represents the pointer for downloading the image.

- Server is represented as S; S={AU,CF,L,G | ϕs}

  Where,

  - AU is automatically updating the system by refreshing the page, connectivity and database.
  - CF stands for client finder. This searches for newly logged in server agents and users associated with the system.
  - L is the centralized log record kept at server.
  - G is the graph section where the graphs for various properties can be seen.

  - A stands for server agent which can be represented as A={AD, IO|ϕa}

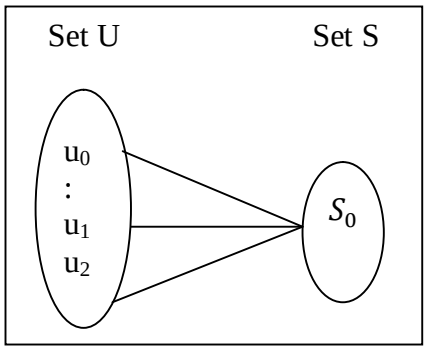
    Where,

  - AD is automatic download; with this the system can download the image automatically, whenever the task is assigned to it.
  - IO is the image operation which will perform different image operations for the user
  - Rq stands for request. This will be transferred from user to server and server to agent.
  - Rs stands for response given by agent to server and server to user.

B. *Activity:*

For each user U there is a server S.

$f_u(u_0)$ |-> $\{S_0\} \epsilon S$



Each server will have definitely produce output for each of the users.

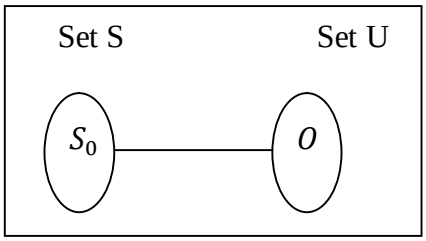
$Fs(s_0)$ |->$\{DI\} \epsilon U$

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

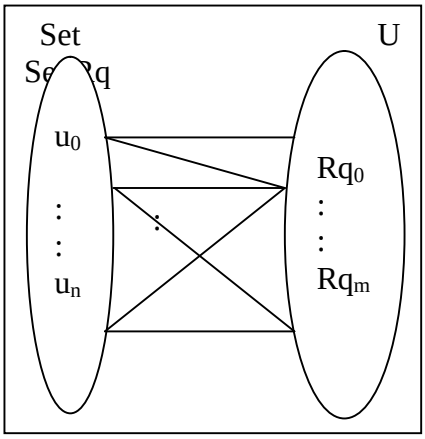
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

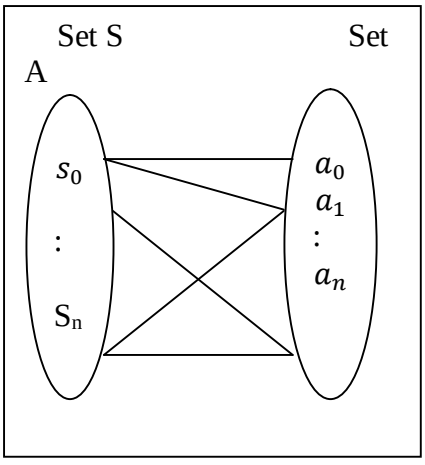
Vol. 3, Issue 7, July 2015



System will have number of users who can be linked with multiple requests R.



One server S can communicate with multiple server agents A.



For each U, S, A etc. a common log L will be maintained.

c. Function Execution time

Registration	1 time
Login	n times
Request	n times

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

Response	n times
Server	n times
Server Agents	n times

d. Efficiency factor
The total performance could be improved by carefully managing the factors, like disciplined user I/O operations, interconnection networks, proficient schedulers, data analysis and memory management.

e. Result through process

If (invalid input)
Cancel
Else
Appropriate result will be shown

B. Algorithm:

a. User- Server communication:

1. User registers to the system.
2. If(Registration successful)
   a. Go to login page.
   b. Show success message.
3. Else
   a. Load registration page again.
   b. Show error message.
4. User attempts for login
5. If(Login successful)
   a. Go to user home page.
   b. Show success message.
6. Else
   a. Load login page again.
   b. Show error message.
7. In Upload image tab,
   a. Select the image/video file to upload.
   b. Select operation to perform.
8. Send request to server.
9. Server searches for available agent to perform operation.
10. Server updates logs and graphs.
11. User can download the results from download image tab.
12. If (another request)
    a. Go to step 7
13. Else
    a. Click on log out button.

b. Server – agent communication:

1. User registers to the system.
2. If(Registration successful)
   a. Go to login page.
   b. Show success message.
3. Else
   a. Load registration page again.
   b. Show error message.
4. User attempts for login
5. If(Login successful)
   a. Go to user home page.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

b. Show success message.

6. Else
   a. Load login page again.
   b. Show error message.
7. Server will search for any active server agent.
8. Server agent waits for assignment of job.
9. If (job assigned)
   a. Agent performs given operation on given image.
   b. Saves the result to particular location.
10. Else
    a. Go to step 8.
11. Server will send link of resulting image to the user.
12. User can download the result.
13. If(more requests)
    a. Go to step 7.
14. Else
15. Logout server agent.
16. Logout server.

C. Working:

In this system, three modules are developed, like user module, server module and server agent module. All these modules work for an image processing application. The images are sent by the user module to server. And server will assign the right to perform operation to a specific server agent. The working of these modules is as follows.

The user first has to register to the system before using it. After registration it can use the system any time by logging in to the system. Server must be active for all the run time of the system. After login successfully, the user may upload the image to the system for performing any of the provided operations. This is sent in the form of <image,operation> tuple to the server.

Now, at the agent side, there may be one or more server agents present as the slave nodes of the system. So they too have to register for the very first time and can log in for all the next attempts.

Whenever the user sends any request to the server, the server will check for logged in server agent and assigns the job to it. If it does not get any active agent, it will hold the request, and keep checking the availability. The server will also do the load balancing of jobs between different active server agents and schedule the jobs accordingly.

Now, the server agent will download the image and perform specified action on it. This result is then updated on the server and it can now be seen by the user.

For all this time, the server will keep logs of these operations, and generate the graphs of throughputs, delays, precision and recall.

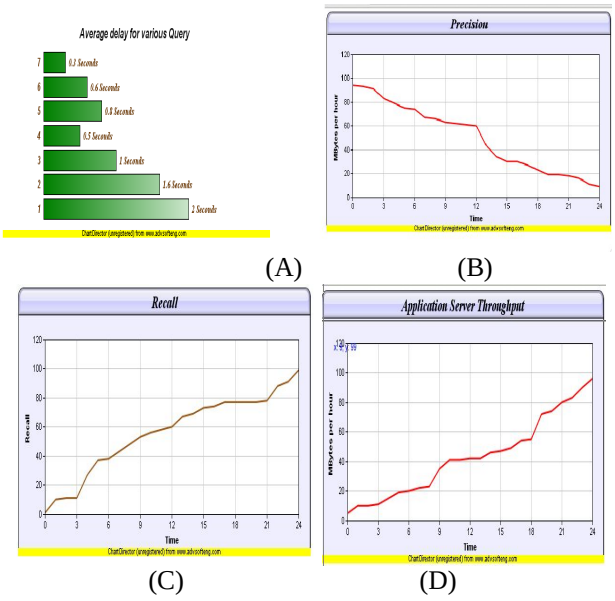
ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

IV. RESULTS



(A) (B)

(C) (D)

Figure 1. Results taken by the server based on the logs

The experiments were conducted on different machines for users, server and server agent modules. Each of these were expected to be of minimal hardware and software requirements.The graphs in the above diagram show the results of system taken by the server module. The figure 1(A) shows the delays of various requests. It may vary for different requests. Figure 1(B) shows precision graph for accuracy of the operations per request in form of speed per available time. It decreases with increase in time. As it is real time system, if fast interactions are done, then system works properly. It does not tolerate the delayed communications. In figure 1(C) the recall of the system is given per time taken by system, it increases with increase in time. In figure 1(D) overall throughput is given for the system performance, which shows continuously increase in the data processed as increase in time is observed. This output is generated by the parameters and time constraints considered since a user registers to the system till whole the operation is done for all logged in users. This checks for all the log records of the system and gives the performance measure for complete product.

V. CONCLUSION

The use of MapReduce is very beneficial for 'Big data'. Especially it is most efficient with the systems like Hadoop. The MapReduce can provide better results by providing it with some extra functionality.

One of the ways to achieve this is used in this system. Pipelining between different phases of MapReduce is used as the base system. This system can be enhanced with some memory scalability and topology aware and high bandwidth network channels for even better results.

Now-a-days the use of this system especially Hadoop is getting increased. Functionality of existing systems can be enhanced with the help of this proposed system. I hope that it will be helpful to industry as well as educational uses of MapReduce.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 7, July 2015

REFERENCES

1]Weikuan Yu, Member, IEEE, Yandong Wang, and Xinyu Que, "Design and Evaluation of Network-Levitated Merge for Hadoop Acceleration," IEEE Transactions On Parallel And Distributed Systems, Vol. 25, No. 3, March 2014
2]J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Proc. Sixth Symp. Operating System Design and Implementation (OSDI '04), pp. 137-150, Dec. 2004.
3] D. Jiang, B.C. Ooi, L. Shi, and S. Wu, "The Performance of MapReduce: An In-Depth Study," Proc. VLDB Endowment, vol. 3, no. 1, pp. 472-483, 2010.
4] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce Online," Proc. Seventh USENIX Symp. Networked Systems Design and Implementation (NSDI), pp. 312-328, Apr. 2010.
5] M. Zaharia, A. Konwinski, A.D. Joseph, R.H. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," Proc. Eighth USENIX Symp. Operating Systems Design and Implementation (OSDI '08), Dec. 2008.
6] Y. Chen, S. Alspaugh, and R.H. Katz, "Interactive Query Processing in Big Data Systems: A Cross Industry Study of MapReduce Workloads," Technical Report UCB/EECS-2012-37,
EECS Dept., Univ. of California, Berkeley, Apr. 2012.
7] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-Network Aggregation for Big Data Applications," Proc. Ninth USENIX Conf. Networked Systems Design and Implementation (NSDI '12), p. 3, 2012.
8] C. Ranger, R. Raghuraman, A. Penmetsa, G.R. Bradski, and C. Kozyrakis, "Evaluating MapReduce for Multi-Core and Multiprocessor Systems," Proc. IEEE 13th Int'l Symp. High Performance Computer Architecture (HPCA '07), pp. 13-24, 2007.
9] Y. Mao, R. Morris, and F. Kaashoek, "Optimizing MapReduce for Multicore Architectures," Technical Report MIT-CSAIL-TR-2010-020, Massachusetts Inst. of Technology, May 2010.
10] R. Chen, H. Chen, and B. Zang, "Tiled-MapReduce: Optimizing Resource Usages of Data-Parallel Applications on Multicore with Tiling," Proc. 19th Int'l Conf. Parallel Architectures and Compilation Techniques (PACT '10), pp. 523-534, 2010.
11] S. Babu, "Towards Automatic Optimization of MapReduce Programs," Proc. First ACM Symp. Cloud Computing (SoCC '10), pp. 137-142, 2010.
12] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Locality- Aware Resource Allocation for MapReduce in a Cloud," Proc. Conf. High Performance Computing Networking, Storage and Analysis, pp. 58:1-58:11, Nov. 2011.
13] H. Herodotou and S. Babu, "Profiling,What-If Analysis, and Cost-Based Optimization of MapReduce Programs," Proc. 37th Int'l Conf. Very Large Data Bases, 2011.
14] G. Ananthanarayanan, S. Kandula, A.G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the Outliers in Map-Reduce Clusters Using Mantri," Proc. Ninth USENIX Symp. Operating Systems Design and Implementation (OSDI '10), pp. 265-278, Oct. 2010.
15] G. Ananthanarayanan, S. Agarwal, S. Kandula, A.G. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters," Proc. Sixth European Conf. Computer Systems (EuroSys '11), Apr. 2011.
16] E. Jahani, M.J. Cafarella, and C. Re, "Automatic Optimization for MapReduce Programs," Proc. VLDB Endowment, vol. 4, pp. 385- 396, 2011.
17] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy, "A Platform for Scalable One-Pass Analytics Using MapReduce," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '11), pp. 985- 996, 2011.
18] J. Liu, J. Wu, and D.K. Panda, "High Performance RDMA-Based MPI Implementation over InfiniBand," Int'l J. Parallel Programming, vol. 32, pp. 167-198, 2004.
19] S. Sur, H. Wang, J. Huang, X. Ouyang, and D.K. Panda, "Can High-Performance Interconnects Benefit Hadoop Distributed File System?" Proc. Workshop Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction
20] Madhura Sadashiv Dhekane, Prof. Megha Borole, "Efficient Topology Aware System of Network Levitated Merge for Hadoop Acceleration", International Journal of Emerging Technology and Advanced Engineering, Volume 5, Issue 3, March 2015.