Volume 5, No. 5, May-June 2014

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Competent Load Rebalancing For Distributed File Systems In Cloud

M.Bala Ganesh[1], P.Sankari[2]
Associate Professor /Department of CSE[1], PG Scholar /Department of CSE[2]
[1,2]Sembodai Rukmani Varatharajan Engg College
Vedaraniam, India
Affiliated by Anna University, Chennai, India

Abstract— Distributed file systems are fundamental factors for cloud computing applications using MapReduce Technique [1]. MapReduce algorithms are used to do searching, sorting and other operations in efficient and parallel way [2]. In distributed file systems, nodes are used for storage and computing functions. Usually file is divided into n number of chunks and chunks will be allocated to n number of nodes in cloud environment. So that MapReduce can perform between the nodes in parallel manner. However, in cloud computing environment failure may occur at any time and nodes may upgrade, add or replaced in the system. Files can also be dynamically deleted, appended and created [1]. This leads to load inequity among the nodes in a distributed system. That is file chunks may not equally distribute across the nodes. Existing distributed file system in clouds implemented based on central load balancer for chunk reallocations [1]. This dependency is completely insufficient in large scale dynamic and data intensive clouds, failure prone environment because of the overload of the central load balancer. CLRDFC against centralized load rebalance technique [1] and strongly recommends distributed load rebalancing algorithm, which taking care of load rebalancing among the nodes. So that load rebalancing task can share across multiple nodes and can avoid total system failure at a time [1]. Replication results indicate that CLRDFC is as good as with the existing central node approach and significantly doing fine than prior distributed algorithm in terms of movement cost, load imbalance factor and algorithmic operational cost.

Keywords- Distributed file systems, Load balancing, Cloud environment , chunk , node

I. INTRODUCTION

Cloud computing is a fast growing technology [1]. The technology behind the distributed file systems are internet and distributed file systems. In Cloud environment users will get storage area and other computer related resources based on their necessity. Our mail services are best examples for cloud computing environments [3]. Just we are having internet access and mail account to access our mails without worrying about storage capacity and hardware requirements. All the mail servers provide storage space for our mail and we are not bothered about hardware and software requirement for mail service, this mail service is a best example for cloud computing [3]. Just we need internet connection and browser to connect our email. Technologies and algorithm behind the cloud computing are MapReduce algorithm, distributed file system, distributed hash table concepts and virtualization[4]. Usually in a cloud computing environment files will be partitioned into n number of chunks and that chunks would be allocated to different nodes called chunk server. This will make Mapreduce tasks easier among different node in a parallel way. For example [1], consider a word occurrences calculation application which counts the number of occurrences of words in a large file. The files are divided as n number of chunks and assigned among the chunk servers. In such a system each chunk server can calculate the word occurrences by scanning the respective chunks in a parallel manner. Uniform distribution of load among different nodes, can drastically increase performance of MapReduce applications [4].

II.RELATED WORK

CLRDFC goal is distribute the file chunks among the nodes as possible equally between different nodes. So that none of the node will handle more number of chunks. Considering geography region of the node greatly reduce the network traffic while load rebalancing [1]. Sometimes nodes may share load between two different very faraway nodes even if there is a possibility to share load with the neighbor node. This may leads to network traffic. Utilizing nodes in efficient way to improve the performance is demanded [1]. CLRDFC recommends splitting the load rebalance task to the individual nodes by having separate DHT's for n number of nodes. Location of a chunk can easily find by simply referring a Distributed Hash Table (DHT) with a key value.

DHT will be having unique key for each and every file chunks in the distributed environment. [1] DHT provides self node management and re adjust mechanism while there is an imbalance in the load due to addition and removal of chunks from the node. Implementation of load rebalance has done by influencing DHT's algorithm for distributed file chunks to improve the uniform balance of the node as much as possible. So that nodes are able to handling load rebalance spontaneously without global knowledge and synchronization. All other existing systems are based on centralized node approach to manage metadata of files and balancing loads of the nodes based on the metadata [2].

When the size of the data increased and number of frequent access to the files increase, central node approach may leads to underperformance due to load congestion problem [1]. In a large scale cloud central approach may lead to load imbalance and total failure of entire system. In a large scale environment cloud can have hundreds of thousands of nodes. [1] Existing load rebalancing DHT algorithms implemented without consideration of physical region of the node and heterogeneity of the node. All DHT's are helping to balance the node generally by maintain node details and adjusting load between the respective nearest nodes [1]. First, the typical unsystematic selection of node to store data leads to load inequity among nodes. Some nodes may end up with a huge load data and some may end up

with very less load of data. An important factor among DHTs is load-balance among the nodes with even distribution of items (or equal load share) to nodes in the DHT. [1] By leveraging the geographical position of node information and also utilizing the capable nodes help us to improve the overall system performance and additionally reduce the complication of DHT algorithm as much as possible [1]. Simulation of the results indicates, proposal is as good as than the existing central node approach and significantly performs fine than prior distributed algorithm in terms of data transportation cost, load imbalance factor and algorithmic operational cost [1].

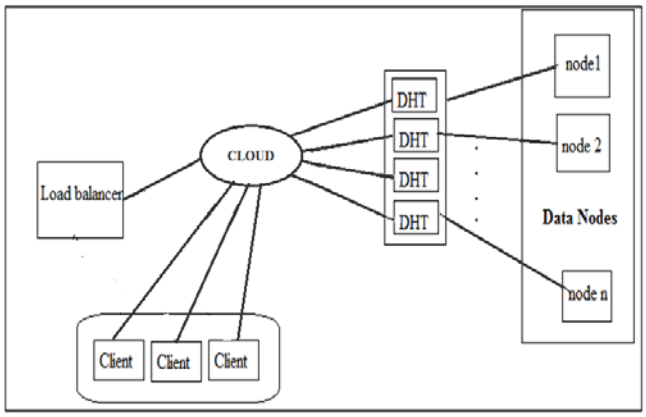


Figure 1. Architecture Diagram of CLRDFC

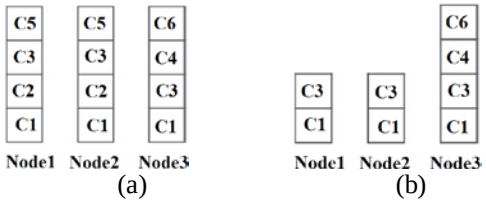
The above architecture diagram [1] demonstrates the distributed DHT's instead centralized DHT approach to maintain well balanced load across the nodes.

III.PROPOSED WORK

In [1] the proposed system distributed load rebalancing algorithm taking care of load rebalance among the nodes. So that load rebalancing task can share across multiple nodes and can avoid dependency on single node. Decentralized [5] DHT algorithm is compared against a centralized approach in a production system and a better than any other existing solution in the cloud environment in terms of load balance and performance factor. Proposed work strongly recommends distributed hash table nodes against centralized node and also considering other key factors like physical location of the node and algorithmic overhead [1].

A. Balancing Chunks:

For example [1], consider set of nodes V is present in the distributed environment and the cardinality of the V is |V| = n. typically n can be hundred or thousand or whatever maybe. In system files are stored in a number of nodes. Set of files represented as F. Each [1] file f € F is split into number of same size chunks represent by Cf. Usually load of the chunks proportional to the number of chunks hosted by the server.



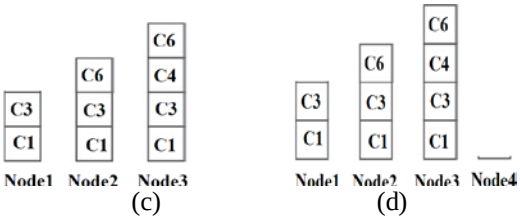


Figure 2. Load rebalancing problem illustration

The above figure example [1] illustrates the load rebalancing problem, where (a) an initially allotment of chunks of 6 files C1, C2, C3, C4, C5, and C6 in three nodes Node1, Node2, and Node3, (b) files C2 and C5 are deleted, (c) C6 is appended, and (d) node Node4 joins. The nodes in (b), (c), and (d) are not in a load balanced state. However, [1] in cloud computing environment node failure may occur at any time and nodes may added, upgrade or replaced in the system. Files [1] can also be dynamically deleted, appended and created. So files in the F dynamically created deleted and modified. The result of these operations leads to load imbalance problem in a system. Objective here is implementing algorithm to rebalance a load (i.e. distributing chunks equivalently as much as possible among the nodes). Transport cost is nothing but migrating a chunks to different locations to balance the load between the nodes.

B. Chunk Rebalancing Factors & Principle:

CLRDFC proposed n number of DHT instead of one centralized DHT to avoid bottleneck problem [1]. Generally a file will be split into n number of equal size chunks and each chunk has a unique identifier generated by SHA1 [4] hash function algorithm. So in DHT nodes details are stored as key and respective physical address of a node is stored as value. So that DHTs can have all the details about the node. DHTs [6] will have details about all the nodes and chunks present in the nodes and respective load details of nodes. dynamic [1] addition, modification and deletion of chunks lead to load imbalance and bring down performance of MapReduce tasks in a large cloud environment. So [8] whenever load imbalance is happening beyond the threshold level, load balancing task must be taken place to manage effective cloud operations.

The new proposal recommends load balance among the same region nodes instead of faraway nodes, it will drastically reduce the load transfer cost while load rebalancing. So each and every time DHT [1] will search for a nearest capable node to share the load within their limited knowledge of DHT table. Suppose if DHT does not find a capable node within their limited entries of key value pairs then it will send random adhoc request to the nearest DHTs to get capable nodes to share load. Once if it gets responses from the DHTs then it will decide and choose the capable nodes to share load based on their geographical region, distance, node capacity and chunk size. Heterogeneity [7] of the nodes (i.e. different nature of the nodes) also important considerable factor while load rebalance. Whenever load imbalance occurs DHTs will try to balance the load within their limited knowledge. Suppose if it does not able to accommodate among the nodes which comes under same DHT then it will try to accommodate the load in balanced state by sharing load with nearest capable nodes. DHTs [2] will find nearest capable nodes by sending random request to the nodes which present under same region and then based on the response, it will decide the capable nodes.

Capable node selections are always based on node heterogeneity [1] , geographical location of the nodes and node capacity and capability. All the above factors made load rebalance as very effective. In large scale cloud environment if the loads are evenly distributed among nodes then MapReduce [9] tasks can perform very effectively and performance of the cloud also will be very high. The nodes used to run load rebalancing algorithm periodically, to maintain the balanced state of the nodes. Load rebalance has achieved without global awareness of DHT [1] and it would greatly reduce the network traffic and movement cost.

C. Utilizing Network Addresses:

In an existing [1] system physical locality of the nodes did not considered as factor, So that there may be a chance of migrating chunks between two faraway regions is very high even though there is possibility of sharing loads among the same region nodes. It leads to unnecessary network traffic and performance problem sometimes. Consideration of geographical [1] location of the nodes before it starts sharing the load among nodes is very effective. Same region nodes should have highest probability to share load among them. Always same region node would give highest priority to share the load. Message round-trip delay method has been used to find nearest node. Physical locality called as network address maintained in the DHT [1] along with chunk server Id and status. Usually nodes from the same geographical locations named with the numerically nearby Id's. For example [1] n, n+1, n+2 and so on. So that nearest node can easily choose by seeing naming convention. Nodes present in the distributed system are may heterogeneous(i.e. number of file chunks can occupy by each node may vary due to heterogeneity of the nodes). Each nodes average capacity of the load in a balanced state described as average threshold level of the node. Typically [1] load of the node is proportional to the file chunks counts in the node.

D. Replica Management:

Generally in distributed file systems the same file chunks may be replicated in different nodes to handle node failure [1] and to improve availability of data. In existing system backup are not managed properly and two three backup of same chunks may placed under the same node. Replicas [10] are used to serve faster and better way for most frequently accessed data. Each light weight node may samples the number of nodes and each node having probability of 1/n to share their loads. Here n is the number of nodes. Consider n replicas for each chunk where the chance of n replicas [1] is stored in the same node due to transfer of chunks is independent of their starting location. For example [1] in the file system with thousand nodes and n =3 (that is 3 replicas for single node) then the probability of storing two replica in the same node is 1/1000000 and storing three replicas in the same node is 1/1000000000. For Example [1] Directory-based scheme has been used to find the k replicas in a distributed system. Usually file chunk is mingled with n-1 pointers that keep track of the n-1 replicas to store in randomly selected chunk servers. For Example [1] redundant replicas in a system in this approach are comparatively very less than existing approaches. Statistically [1] for replica n = 2, n = 4 there is no redundant replicas in a node but for the case n =8, there would be more possibility of redundant node. Statistically 2% of the nodes may have redundant replicas for n=8.

IV. PERFORMANCE EVALUATION

In Distributed environment loads of the nodes are rebalanced spontaneously [1] by the nodes. Which greatly decrease the time and increase the performance due to this reliability and scalability [4] and performance of the system would be significantly enhanced. Localities of the chunks are physical address [1] of the chunks which stored. Finding region address of the chunks to share loads among nearest nodes can greatly reduce the traffic of the network and increase the performance of the cloud. Heterogeneity [2] of the network deals and considers the factor of different chunks sizes in the network also used to avoid unnecessary network traffic and making enough space to increase the network performance. Reproduction system is nothing keeping backup copies of chunks to handle failure of nodes and sometimes high frequently accessed data would keep in different places for fast serving purposes.

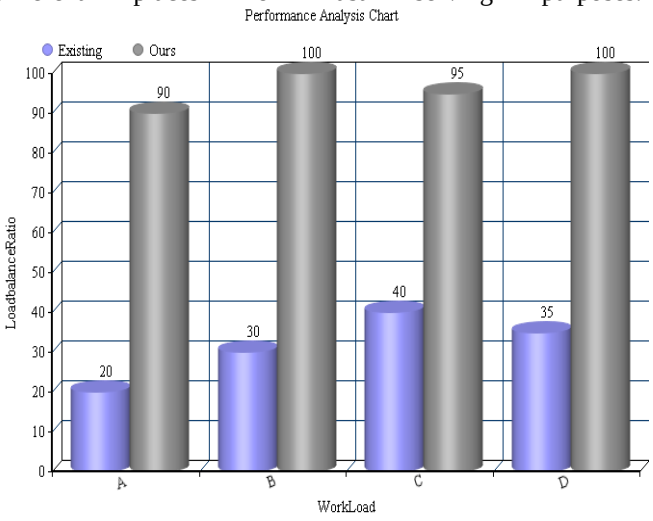


Figure. 3. Load distribution chart

Based on above Load distribution chart workload of the nodes evenly distributed across the multiple nodes and its leads to increase the performance of Map reduce tasks in the cloud.

The above simulated chart explains and compares how effectively load has been rebalanced in our system than existing load balancing system. Simulation results prove the effectiveness of the distributed load rebalancing concepts. The simulation result clearly shows the effectiveness of the proposed system compare with existing system.

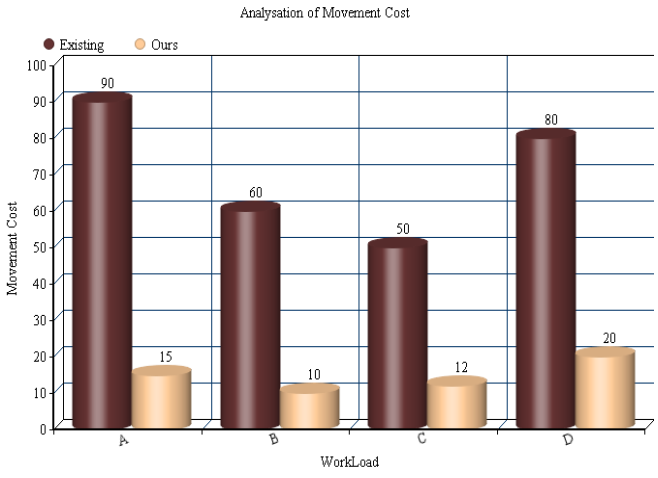


Figure. 4. Movement cost chart

Comparison of load movement cost has been demonstrated in the above chart. Simulation results noticeably shows the effectiveness of considering geographical location of the node while load rebalancing.

V. CONCLUSION

Based on the study and conclusion our approach for the load rebalance through distributed hashtable among the nodes delivering very high performance and well balanced node maintenance. Consideration of the geographical location [1] of a node greatly reduces the movement cost and network traffic. This approach leads to balance the masses of nodes and bring down the movement cost [1], whereas taking advantage of physical network environs and node non uniformity (heterogeneity). Load rebalance algorithm [1] and Replica management concepts are greatly enhanced and significantly performs well than existing system.

VI. REFERENCES

[1]. Hung-Chang Hsiao,Hsueh-Yi Chung, Haiying Shen and Yu-Chang Chao, "Load Rebalancing for Distributed File Systems in Clouds" , Proc. IEEE Transactions on parallel and distributed systems, Vol. 24, No. 5, May 2013

[2]. Bharambe A, Agrawal A and Seshan (2004) 'Mercury: Supporting Scalable Multi Attribute Range Queries' Proc. ACM SIGCOMM '04, pp. 353-366.

[3]. Eastlake D and Jones P (2001) 'US Secure Hash Algorithm 1 (SHA1),' RFC 3174, Sept. 2001.

[4]. Ganesan P, Bawa M and Garcia-Molina H (2004) 'Online Balancing of Range Partitioned Data with applications' Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04),pp.444-455, Sept 2004

[5]. Ghemawat S, Gobioff H and Leung (2003) 'The Google File System,' Proc. 19th ACM Symp. Operating Systems Principles (SOSP'03), pp. 29-43, Oct. 2003.

[6]. Gudadhe P.D, Gawande A.D and Gautham L.K (2009) 'Enhance the performance of Hadoop distributed file system for random file access using increased block size'. Proc. ACM SIGCOMM '09, pp. 63-74,.

[7]. Hairong Kuang and Robert Chansler 2012) 'Enhanced Hadoop Distributed File Sys' vol. 63, no. 6, pp. 217-240, Mar. 2012

[8]. Manku G.S (2004) 'Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables,' Proc. 23rd ACM Symp. Principles Distributed Computing (PODC '04), pp. 197-205, July 2004.

[9]. Rowstron A and Druschel P (2001) 'Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems,' Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms Heidelberg, pp. 161-172, Nov. 2001.

[10]. John Howard, Michael Kazar, Sherri Menees, Robert Sidebotham, and Michael West. Scale and performance in a distributed file system. ACM Transactions on Computer Sys, 6(1):51–81, Feb 1988.