Vol.2, No.2 ISSN Number (online): 2454-9614

A Freight Equilibrium Model Based on Cloud Disproportion for the Shared Cloud

Asha Sk. #1, Om Prakash Samantray#2, Y.chandana#3

1M.Tech Student, Department of CSE, Narsaraopet Engineering College, Narsaraopet, India.
2Associate Professor, NEC, Narasaraopet
3Assistant Professor, NEC, Narasaraopet
1ashashaik31@gmail.com
2om.prakash02420@gmail.com
3chandu.yamani@gmail.com

***Abstract*—Cloud computing having tremendous growth on recent years but it is not segregation on shared clouds. Distributed file systems are key building blocks for cloud computing applications based on the Map Reduce programming paradigm. In such file systems, nodes simultaneously serve computing and storage functions; a file is partitioned into a number of chunks allocated in distinct nodes. However, in a cloud computing environment, failure is the norm, and nodes may be upgraded, replaced, and added in the system. Files can also be dynamically created, deleted, and appended. This result in load imbalance, that is, the file chunks are not distributed as uniformly as possible in the nodes. Although shipment equilibrium algorithms exist in the literature to deal with the load imbalance problem. Cloud balancing moves global server shipment equilibrium from traditional routing options based on static data to context-aware distribution across cloud-based services, both internal and external.**

I. INTRODUCTION

Cloud Computing is a technology, which connects so many nodes together for allocating resources dynamically. Different types of technologies are used in clouds such as Map Reduce programming paradigm, distributed file systems, virtualization. These kinds of techniques are scalable which can add or delete new nodes or systems making it reliable. In a large cloud we can add thousands of nodes together. The main aim is to allocate files to these nodes without making heavy load to any of the nodes, for that files are partitioned into different modules. Another objective is to reduce the network inconsistencies and network traffic because of the unbalancing of loads. The reduction of network inconsistency will lead to maximization of network bandwidth so that so many large applications can run in it. Due to scalability property we can add, delete and update new nodes so that it supports heterogeneity of the system. To improve the capability of nodes we use Distributed file System in Cloud Computing Applications. In such file systems the main functionalities of nodes is to serve computing and storage functions. If we want to store a file into the system firstly we will divide the file into different modules and store it in different nodes. So we introduced new load rebalancing algorithm to avoid all these disadvantages. When analysing the existing system clouds rely on central nodes to balance the loads of storage nodes, there comes the performance bottleneck because the failure of central nodes leads to the failure of whole system.

Distributed file systems are key building blocks for cloud computing applications based on the Map Reduce programming paradigm. In such file systems, nodes simultaneously serve computing and storage functions; a file is partitioned into a number of chunks allocated in distinct nodes so that Map Reduce tasks can be performed in parallel over the nodes. For example, consider a wordcount application that counts the number of distinct words and the frequency of each unique word in a large file. In such an application, a cloud partitions the file into a large number of disjointed and fixed-size pieces (or file chunks) and assigns them to different cloud storage nodes (i.e., chunk servers). Each storage node (or node for short) then calculates the frequency of each unique word by scanning and parsing its local file chunks.

In such a distributed file system, the load of a node is typically proportional to the number of file chunks the node possesses [3]. Because the files in a cloud can be arbitrarily created, deleted, and appended, and nodes can be upgraded, replaced and added in the file system [7], the file chunks are not distributed as uniformly as possible among the nodes. Load balance among storage nodes is a critical function in clouds. In a load-balanced cloud, the resources can be well utilized and provisioned, maximizing the performance of MapReduce-based applications.

II. RELATED WORK

There have been many studies of freight stabilizing for the cloud environment. Load stabilizing in cloud computing was described in a white paper written by Adler who introduced the tools and techniques commonly used for load stabilizing in the cloud. However, load stabilizing in the cloud is still a new problem that needs new architectures to adapt to many changes. Chaczko et al. described the role that load balancing plays in improving the performance and maintaining stability.
The underlying concept of cloud computing was introduced way back in 1960s by John McCarthy [4]. His opinion was that "someday, computation may be organized as a public utility" [4]. In 1966 Douglas Parkhill investigated the characteristics of cloud computing in his book, “the Challenge of the Computer Utility” for the first time [4]. The history of the term cloud is from the telecommunications world, where telecom companies started offering Virtual Private Network

(VPN) services [5] with comparable service quality at a lower cost. Initially before VPN, they provided dedicated point-to-point data circuits which were wastage of bandwidth. But by using VPN services, they can switch traffic to balance the utilization of overall network. Cloud computing now extends this to concept to cover servers and network infrastructure.

Load-balancing algorithms based on DHTs have been extensively studied (e.g., [13], [14]). However, most existing solutions are designed without considering both movement cost and node heterogeneity and may introduce significant maintenance network traffic to the DHTs. In contrast, our proposal not only takes advantage of physical network locality in the reallocation of file chunks to reduce the movement cost but also exploits capable nodes to improve the overall system performance. Additionally, our Algorithm reduces algorithmic overhead introduced to the DHTs as much as possible.

State-of-the-art distributed file systems (e.g., Google GFS [2] and Hadoop HDFS [3]) in clouds rely on central nodes to manage the metadata information of the file systems and to balance the loads of storage nodes based on that metadata. The centralized approach simplifies the design and implementation of a distributed file system. However, recent experience (e.g., [8]) concludes that when the number of storage nodes, the number of files and the number of accesses to files increase linearly, the central nodes (e.g., the master in Google GFS) become a performance bottleneck, as they are unable to accommodate a large number of file accesses due to clients and MapReduce applications. Thus, depending on the central nodes to tackle the load imbalance Problem exacerbate their heavy loads. Even with the latest development in distributed file systems, the central nodes may still be overloaded. For example, HDFS federation [9] suggests architecture with multiple namenodes (i.e., the nodes managing the metadata information). Its file system namespace is statically and manually partitioned to a number of namenodes. However, as the workload experienced by the namenodes may change over time and no adaptive workload consolidation and/or migration scheme is offered to balance the loads among the namenodes, any of the namenodes may become the performance bottleneck.

In this paper, we are interested in studying the load rebalancing problem in distributed file systems specialized for large-scale, dynamic and data-intensive clouds. Our objective is to allocate the chunks of files as uniformly as possible among the nodes such that no node manages an excessive number of chunks. Additionally, we aim to reduce network traffic (or movement cost) caused by rebalancing the loads of nodes as much as possible to maximize the network bandwidth available to normal applications.

III. EXISTING SYSTEM

Node failure is the norm in aforementioned distributed system, and the module servers may be upgraded, replaced and added in the system. Let, the set of files as F. Files in F may be arbitrarily created, deleted, and appended. Considering a large-scale distributed file system consisting of a set of module servers M in a cloud. Each file f is partitioned into a number of disjointed, fixed-size modules. Existing solutions to balance load in DHTs incur a high overhead either in terms of routing state or in terms of load movement generated by nodes arriving or departing the system. All DHTs make some effort to load balance, generally by (i) randomizing the DHT address associated with each item with a good enough hash function and (ii) making each DHT node responsible for a balanced portion of the DHT address space. First, the typical random partition of the address space among nodes is not completely balanced. Some nodes end up with a larger portion of the addresses and thus receive a larger portion of the randomly distributed items. An important issue in DHTs is load-balance the even distribution of items (or other load measures) to nodes in the DHT. This results the uneven distribution of module servers.

Limitations of Existing System

Emerging distributed file systems depends on a central node for module reallocation. This dependence is inefficient in an exceedingly large-scale, failure prone atmosphere as a result of the central load balancer is tested underneath sure employment that's linearly scaled with the system size, and will so become the performance bottleneck and therefore the single purpose of failure.

IV. PROPOSED SYSTEM

In one vision of the future, the shifting of load is automated to enable organizations to configure clouds and cloud balancing and then turn their attention to other issues, trusting that the infrastructure will perform as designed. This future is not so far away as it may appear.

Algorithm: Best Partition Searching

```
1. Begin
2. While job do
3. SearchBestPartition (job);
4. If partition State == idle || partition State == normal
   then
5. Send Job to Partition;
6. End if
7. Else
8. Search for another Partition;
9. End while
10. End
```

Implementation of data distributer sharing system

The Round Robin algorithm is one of the simplest load harmonizing algorithms, which passes each new request to the next server in the queue. The algorithm does not record the status of each connection so it has no status information. In the regular Round Robin algorithm, every node has an equal opportunity to be chosen. However, in a public cloud, the configuration and the performance of each node will be not the same; thus, this method may overload some nodes. Thus, an improved Round Robin algorithm is used, which called “Round Robin based on the load degree evaluation”. The algorithm is still fairly simple. Before the Round Robin step, the nodes in the load harmonizing table are ordered based on the load degree from the lowest to the highest. The system builds a circular queue and walks through the queue again and again. Jobs will then be assigned to nodes with low load degrees. So we are using shipment equilibrium is the process of reassigning the total load to the individual nodes of the collective system to make effective resource utilization and to improve the response time of the jobs, simultaneously removing a condition in which some of the nodes are over loaded while some others are under loaded. Depending on system state, shipment equilibrium algorithms divided into two types as static and dynamic. A shipment equilibrium algorithm which is dynamic in nature, does not consider previous state or behaviour of the system, that is, it depends on the present behaviour of the system. Depending on who initiated the process, shipment equilibrium algorithms can be divided into three types as sender Initiated, receiver Initiated and symmetric. The Classification of shipment equilibrium algorithms is shown in figure 1.

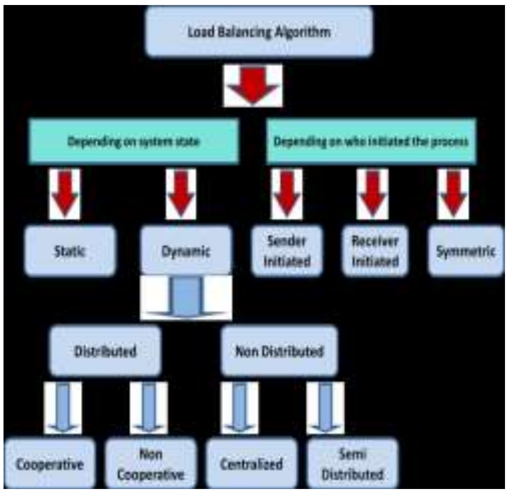

Fig 1: Classification of shipment equilibrium algorithms

Then used Dynamic shipment equilibrium can be categorized in two types as distributed and non-distributed. In distributed algorithms all nodes present in the system execute the algorithm and the task of shipment equilibrium is shared among them [7]. Nodes interaction can take two forms: cooperative and non-cooperative [13]. In the cooperative, the nodes work side-by-side to achieve a common objective to improve the overall response time, etc. In the non-cooperative, each node works independently toward a goal to improve the response time of a local task. In non-distributed type, either one node or a group of nodes do the task of shipment equilibrium. Non-distributed dynamic shipment equilibrium algorithms can take two forms: centralized and semi-distributed. In the first form, the shipment equilibrium algorithm is executed only by a single node in the whole system: the central node. In semi-distributed form, nodes of the system are partitioned into clusters, where the shipment equilibrium in each cluster is of centralized form. A central node is elected in each cluster by appropriate election technique which takes care of shipment equilibrium within that cluster.

V. PERFORMENCE EVALUATION

The performance of our algorithm is evaluated through computer simulations. Our simulator is implemented with PThreads. In the simulations, we carry out our proposal based on the Chord DHT protocol [10] and the gossip-based aggregation protocol in [16] and [17]. In the default setting the number of nodes in the system is n =1000, and the number of file chunks is m =10,000. To the best of our knowledge, there are no representative realistic workloads available. Thus, the number of file chunks initially hosted by a node in our simulations follows the geometric distribution enabling stress tests as suggested [15] for various load rebalancing algorithms. It shows the cumulative distribution functions (CDF) of the file chunks in the simulations, where workloads A, B, C, and D represent four distinct geometric distributions. Specifically, these distributions indicate that a small number of nodes initially possess a large number of chunks. The four workloads exhibit different variations of the geometric distribution.

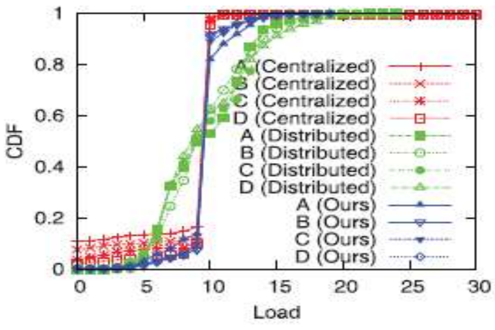

Fig 2: The Load Distribution

Fig 2 represents the simulation results of the load distribution after performing the investigated load-balancing algorithms. Here, the nodes simulated have identical capacity. The simulation results show that centralized matching performs very well as the load balancer gathers the global information from the namenode managing the entire file System. Since A ¼ 10 is the ideal number of file chunks a node should manage in a load-balanced state, in centralized matching, most nodes have 10 chunks. In contrast, distributed matching performs worse than centralized matching and our proposal. This is because each node randomly probes other nodes without global knowledge about the system. Although our proposal is distributed and need not require each node to obtain global system knowledge, it is comparable with centralized matching and remarkably outperforms distributed matching in terms of load imbalance factor.

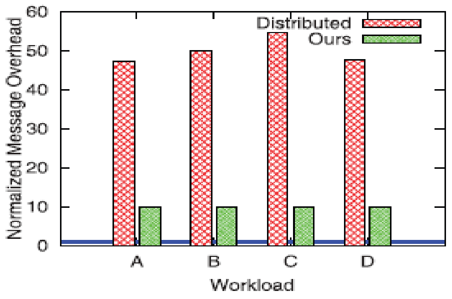

Fig 3: The Message Overhead

Fig 3 shows the total number of messages generated by a Load rebalancing algorithm, where the message overheads in distributed matching and our proposal are normalized to that of centralized matching. The simulation results indicate that centralized matching introduces much less message overhead than distributed matching and our proposal, as each node in centralized matching simply informs the centralized load balancer of its load and capacity. On the contrary, in distributed matching and our proposal, each node probes a number of existing nodes in the system, and may then reallocate its load from/to the probed nodes, introducing more messages. We also see that our proposal clearly produces fewer messages overhead than distributed computing. Specifically, any node i in our proposal gathers partial system knowledge from its neighbors whereas node i in distributed matching takes O (log n) messages to probe a randomly selected node in the network.

We considered a large-scale distributed file system consisting of a set of chunk servers V in a cloud, where the cardinality of V is jVj ¼ n. typically, n can be 1,000, 10,000, or more. In the system, a number of files are stored in the n chunk servers. First, let us denote the set of files as F. Each file f 2 F is partitioned into a number of disjointed, fixed size chunks denoted by Cf. For example, each chunk has the same size, 64 Mbytes, in Hadoop HDFS [3]. Second, the load of a chunk server is proportional to the number of chunks hosted by the server [3]. Third, node failure is the norm in such a distributed system, and the chunk servers may be upgraded, replaced and added in the system. Finally, the files in F may be arbitrarily created, deleted, and appended. The net effect results in file chunks not being uniformly distributed to the chunk servers. Our objective in the current study is to design a load rebalancing algorithm to reallocate file chunks such that the chunks can be distributed to the system as uniformly as possible while reducing the movement cost as much as possible. Here, the movement cost is defined as the number of chunks migrated to balance the loads of the chunk servers. Our proposal clearly outperforms the HDFS load balancer. When the name node is heavily loaded our proposal remarkably performs better than the HDFS load balancer. For example, if M¼1%, the HDFS loads balancer takes approximately60 minutes to balance the loads of data nodes. By contrast, our proposal takes nearly 20 minutes in the case ofM¼1%. Specifically, unlike the HDFS load balancer, our proposal is independent of the load of the namenode. It further shows the distributions of chunks after performing the HDFS load balancer and our proposal. As there are 256 file chunks and 25 data nodes, Due to space limitation, we only offer the experimental results for M¼1 and the results for M6¼ 1conclude the similar. Figs. 11c and 11d indicate that our proposal is comparable to the HDFS load balancer, and balances the loads of data nodes, effectively.

A novel load-balancing algorithm to deal with the load rebalancing problem in large-scale, dynamic, distributed file systems in clouds has been presented in this paper. Our proposal strives to balance the loads of nodes and reduce the demanded movement cost as much as possible, while taking advantage of physical network locality and node heterogeneity in the absence of representative real workloads.

VI. CONCLUSION & FUTURE WORK

This paper discuss a new wave of online services in the field of information technology: cloud computing with its challenges. In cloud computing, there are infinite computing capabilities with attractive pay-per-use scheme. Cloud computing provide everything as a service to their users, like as: storage of data as a service, application software as a service, computing platform as a service and computing infrastructure as a service etc. However this wave still needs to resolve some of its existing issues with urgency. One of the major issues of cloud computing is system shipment equilibrium, because overloading of a particular node makes it slow down resulting poor system efficiency. So there is always a requirement of efficient shipment equilibrium algorithms for improving the utilization of computing resource

It aims at having a backup plan in case the system fails even partially. Also work is done to maintain the system stability. There are provisions to accommodate future modifications in the system. Thus, we have successfully gathered information of project and hopefully implement Load Balancing Model for better utilization and performance of cloud services.

REFERENCES

[1] Marios D. Dikaiakos, George Pall is, Dimitrios Katsaros, Pankaj Mehra, Athena Vakali, 2009 "Cloud computing : Distributed Internet Computing for IT and Scientific Research" IEEE Internet Computing, Published by the IEEE Computer Society.

[2] Panagiotis Kalagiakos, Panagiotis Karampelas, 2011 "Cloud Computing Learning" IEEE, Hellenic American University Manchester, N.H. - U.S.A, pp: 7/11.

[3] Peter Mell,Timothy Grance, The NIST Definition of "Cloud Computing" National Institute of Standards and Technology - Computer Security Resource Center-www.csrc.nist.gov.

[4] Cloud Computing,accessed(23/01/2013),from http:// en. Wikipedia .org/ Wiki / Cloud_computing.

[5] John Harauz, Lorti M. Kaufinan. Bruce Potter, 2009 "Data Security in the World of Cloud Computing" IEEE Security & Privacy, Copublished by the IEEE Computer and Reliability Societies

[6] Ramgovind S, Eloff MM, Smith E, 2010 "The management of security in cloud computing" IEEE. [7] Ali M. Alakeel, 2010 "A Guide to Dynamic adbalancing in Distributed Computer Systems" IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.6.

[7] Hadoop Distributed File System “Rebalancing Blocks,” http:// developer.yahoo. com /hadoop /tutorial/module2.html#rebalancing, 2012.

[8] K. McKusick and S. Quinlan, “GFS: Evolution on Fast-Forward,” Comm. ACM, vol. 53, no. 3, pp. 42-49, Jan. 2010.

[9] HDFS Federation, http://hadoop.apache.org/common/docs/ r0.23.0/hadoop-yarn/hadoop-yarn-site/Federation.html, 2012.

[10] I.Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications,” IEEE/ACM Trans. Networking, vol. 11, no. 1, pp. 17-21, Feb. 2003.

[11] A.Rowstron and P. Druschel, “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.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W.Vogels, “Dynamo: Amazon’s Highly Available Key-Value

Store," Proc. 21st ACM Symp. Operating Systems Principles (SOSP '07), pp. 205-220, Oct. 2007.

[13] Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS '02), pp. 68-79, Feb. 2003.

[14] D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," Proc. 16th ACM Symp. Parallel Algorithms and Architectures (SPAA '04), pp. 36-43, June 2004.

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

[16] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-Based Aggregation in Large Dynamic Networks," ACM Trans. Computer Systems, vol. 23, no. 3, pp. 219-252, Aug. 2005.

[17] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M.V. Steen, Gossip-Based Peer Sampling," ACM Trans. Computer Systems, vol. 25, no. 3, Aug. 2007.

Selected from International Conference on Computing (NECICC-2k15)