Volume 5, Issue 1, January 2015 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Load Rebalancing in Public Cloud Based on Best Partitioning Technique by dividing the Cloud

[1]G. Archana Reddy, [2]A. Malla Reddy, [3]Dr. I.Sathyanarayana

1 M.Tech (CSE), Department of Computer Science & Engineering

Sri Indu Institute of Engineering & Technology, Sheriguda (V), Ibrahimpatnam (M), RR Dist, India

[2]Research Scholar (JNTUH), Department of Computer Science & Engineering, Professor & HOD (CSE), India

[3]Principal, Sri Indu Institute of Engineering & Technology, Sheriguda (v), RR District, Hyderabad, India

Abstract: Cloud computing is emerging technology where different users uses the resources dynamically. The number of users using the file systems is increasing day by day. Therefore distributed file systems are building blocks for cloud environment. When a client uploads a file it is partitioned into number of chunks to distinct nodes so that map reduces can be performed in parallel among the nodes. In addition to this, in cloud computing environment failure can occur, nodes can be replaced and/or added in the system. Files can also be deleted or created. Therefore it could result in load imbalance problem. To overcome this problem, a fully distributed load rebalancing algorithm is proposed. Hence the objective is to allocate chunks of files as uniformly as possible among the nodes so that no node manages excessive number of chunks while reducing the movement cost. Each node in system performs the load rebalancing algorithm independently. Each node implements a gossip based aggregation protocol to collect the load status from different nodes. The file chunks which are stored in node get deleted dynamically. We migrate the file chunks to the previous node without randomly selecting node for migration. Thereby we can reduce the movement cost and reallocate the file chunks uniformly among the nodes. This process is repeated until it reaches the last node in the system. This helps disseminate heap of a solitary name node to numerous name nodes. Our second commitment is to utilize a productive convention to send and course information. Our convention can accomplish full connection usage and henceforth diminished download times. Taking into account recreation our convention can outflank HDFS and consequently GFS.

Keywords- Distributed file system, Cloud, Chunk server, Rebalance Cloud architecture, resource allocation,

I. INTRODUCTION

The rapid growth of communication technologies and the Internet in particular has transformed the way we live and work. It has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased. There have been many companies which have plunged deep in this large investment Shows the increasing interest in this new technology.

However this growth comes with increasing and complex challenges of how to transfer compute and store data reliably and in real-time. Some of the challenges include data transfer bottlenecks, performance unpredictability, scalable storage, fast scaling to varying workloads, etc. Dealing with these challenges of large scale distributed data, compute and storage intensive applications such as social networks and search engines requires robust, scalable and efficient algorithms and protocols. These file systems use a name node to keep a list of all files in the cloud and their respective metadata (I-node). Besides the name node has to manage almost all file related operations such as open, copy, move, delete, update, etc. This may not scale and can potentially make the name node a resource bottleneck. Other limitation of this is that the name node is a single point of failure for an HDFS installation. If the name node goes down, the file system is offline. When it comes back up, the name node must replay all outstanding operations. This replay process can take over half an hour for a big cluster.

In Cloud computing environment, disappointment is the standard, and the piece servers may be overhauled, supplanted and included the framework which prompts load awkwardness in the conveyed document frameworks. It implies that the record lumps are not disseminated fairly between the nodes. Conveyed document frameworks in mists, for example, GFS and HDFS, depend on focal servers (expert for GFS and Name Node for HDFS) to deal with the metadata and the heap adjusting. The expert rebalances copies intermittently: information must be moved from an information chunk server V to another on the off chance that its free space is beneath a certain edge. In any case, this brought together approach can incite a bottleneck for those servers as they get to be not able to deal with countless gets to. Therefore, managing the heap awkwardness issue with the focal nodes entangle more the circumstances as it builds their overwhelming burdens. With a specific end goal to oversee expansive number of piece servers to work in cooperation, and take care of the issue of burden adjusting in disseminated document frameworks, there are a few methodologies that have been proposed, for example, reallocating record pieces such that the lumps can be appropriated to the framework as consistently as could be expected under the circumstances while decreasing the development cost however much as could reasonably be expected. Here a completely circulated rebalancing algorithm is proposed to unravel the awkwardness condition of the

nodes in the framework. This algorithm can be incorporated with Hadoop Single node or Multi node Group. Here we have actualized utilizing Single Node Group. In this paper, we acquaint a heap rebalancing algorithm with tackle the heap adjusting issue among all lump servers (i.e. node in short) in the conveyed record framework. We rebalance the heap by moving the pieces to the past node in the framework.

In this paper, we address these problems with current systems such as the GFS/HDFS. In order to make the system scalable, our scheme uses a light weight front-end server to connect all requests with many name nodes. This helps distribute load of a single name node to many name nodes. Our front-end just manages sessions and hence is not a resource bottleneck. Also, our frontend is stateless, therefore if it goes down, no data is lost and bringing it up is very fast. The other feature of our system is that it uses an efficient protocol to send and route data. Our protocol can achieve full link utilization and hence decreased download times. As a result of this, it can achieve lower chunk transfer times and it is much more efficient than HDFS.

The main contributions of this paper include:

- A new distributed architecture with light weight frontend server which is much more scalable than the existing systems such as the HDFS/GFS.
- An efficient protocol to send and route data, which leads to a better link utilization than TCP and hence faster data chunk transfer time.
- A comparative analysis of our protocol with GFS/HDFS.

II. PROJECT METHODOLOGY TO DEVELOP LOAD REBALANCING

An alternate prevalent document framework for organized computer is the Network File System (NFS). It is an approach to impart records between computers on a network as though the documents were found on the customer's nearby hard drive. One of the burdens of NFS is that it tries to make a remote record framework show up as an issue document framework, yet it's perilous to depend on that distortion. There are numerous circumstances in which the utilization of NFS (contrasted with a nearby file system) is not proper or solid. Andrew File System (AFS) is a conveyed arranged record framework which utilizes a set of trusted servers to present a homogeneous, area straightforward record name space to all the customer workstations. AFS has a few advantages over customary organized document frameworks, especially in the ranges of security and adaptability. It is not remarkable for big business AFS cells to surpass twenty five thousand customers. AFS utilizes Kerberos for validation, and executes access control records on catalogs for clients and gatherings. Every customer reserves records on the neighborhood filesystem for expanded speed on resulting appeals for the same document. AFS may not be advantageous for extensive scale document frameworks, for example, the once took care of by GFS.

Frangipani is a scalable distributed file system that manages a collection of disks on multiple machines as a single shared pool of storage. The machines are required to be under a common administrator and be able to communicate securely. It has a very simple internal structure which enables them to handle system recovery, reconfiguration and load balancing very easily. Large-scale distributed system varies inside in different conditions. For example, chunk servers will be added to or withdrew from the system from time to time. Furthermore, a number of performance parameters of the system are always changing. Each of the servers takes the role of either a forager or a scout investigated a distributed and scalable load balancing approach that uses random sampling of the system domain to achieve self-organization thus balancing the load across all nodes of the system. Here a virtual graph is constructed, with the connectivity of each node (a server is treated as a node) representing the load on the server. Each server is symbolized as a node in the graph, with each in degree directed to the free resources of the server. The load balancing scheme used here is fully decentralized, thus making it apt for large network systems like that in a cloud. The performance is degraded with an increase in population diversity.

The main objective of the algorithm is to minimize the system cost by moving the tokens around the system. But in a scalable cloud system agents cannot have the enough information of distributing the work load due to communication bottleneck. So the workload distribution among the agents is not fixed. The drawback of the token routing algorithm can be removed with the help of heuristic approach of token based load balancing. This algorithm provides the fast and efficient routing decision. In this algorithm agent does not need to have an idea of the complete knowledge of their global state and neighbor working load. To make their decision where to pass the token they actually build their own knowledge base. This knowledge base is actually derived from the previously received tokens. So in this approach no communication overhead is generated. ESWLC is an improved form of weighted least-connection (WLC) along with its features, it also taken into account time series and trials. However WLC counts the connections of each server and reports the appropriate server based on the multiplication of a server weight and its count of connections, ESWLC algorithm concludes assigning a certain task to a node only after getting to know about the node capabilities. ESWLC builds the decision based on the experience of the node's CPU power, memory, number of connections and the amount of disk space currently being used. ESWLC then predicts which node is to be selected based on exponential smoothing.

III. OBJECTIVE OF PROJECT

Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system

IV. LOAD REBALANCE PROBLEM

We consider a large-scale distributed file system consisting of a set of *chunk server V* in a cloud, where *V* is |*V* | = *n.* typically *n* can be 1000, 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 F* is partitioned into number of fixed-size chunks denoted by C_f. For example; each chunk has the same size, 64Mbytes, in Hadoop HDFS. Second, the load of a chunk server is proportional to the number of chunks hosted by the server. 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 chunks 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 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. Note that "chunk servers" and "nodes" are interchangeable in this paper.

V. OUR PROTOCOL

The main components of our protocol are the user client (UCL), light weight front end server (FES), and some name node servers (NNS), a resource allocator (RA), block servers (BS) and resource monitors (RM). As shown in Figure 2 users of our file system connect by invoking the UCL. The UCL connects users to the FES. The FES manages sessions with the clients and then forwards the client requests to an NNS. An NNS stores the users file system Meta data and reference to a BS which in turn stores the data blocks of a file. The RA tells the NNS which BS and path to BS to use to store data in the BS based on the resource monitor value (rate) it gets from each RM. An RM associated with each BS monitors the resource at its BS and periodically sends a rate metric to the RA.

The Algorithm

As shown in Figures 2 and 1 our protocol uses the following steps.

step1. A user application initiates a session with FES using a UCL.

step2. The FES authenticates the user request, finds an appropriate NNS for example by hashing the request ID, and sends the name or ID of the NNS (along with the NNS password) back to the user application.

step3. The NNS in turn asks the RA connected to the local switch for an appropriate BS and a path to the BS in which the user application (or another node in the cloud) can store blocks of data or from which it can retrieve the previously stored blocks of data. The RA uses the rate metric it gets from each RM, from itself and other RAs to do the resource allocation. The RA is like a software router. An RA and the network switch can serve as a router. More on how the RA finds the appropriate BS is discussed in section 3.3.

step4. The NNS sends name or ID of the BS to the user application and request ID and password to the BS.

VI. LOAD REBALANCING ALGORITHM

Here we assume the entire node has identical capacity and node can handles equal number of chunks.

step5. The user application requests the BS using the information it got from the NNS to store data or retrieve data blocks.

step6. The BS authenticates the user request using the information it got from the NNS and continues to transfer data to the user or store data from the user.

step7. The RM associated with the BS periodically sends the rate metric which serves as an aggregate resource monitor.

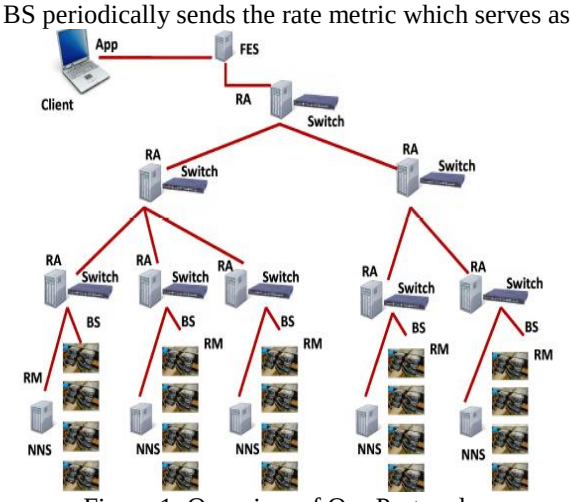

Figure 1: Overview of Our Protocol

F= {f_1, f_2, f3,… f_r}
Ni= {n_1, n2, n3…….n_s}
G= defines capacity of node
Ni= {n_1, n2, n3…….n_s}
G= defines capacity of node

Boolean flag=false
Boolean full=true
m =defines the total number of nodes in the system
s =defines the number of chunks in nodes
b= files splitted into number of chunks based on file size
Ck=defines the number of chunks

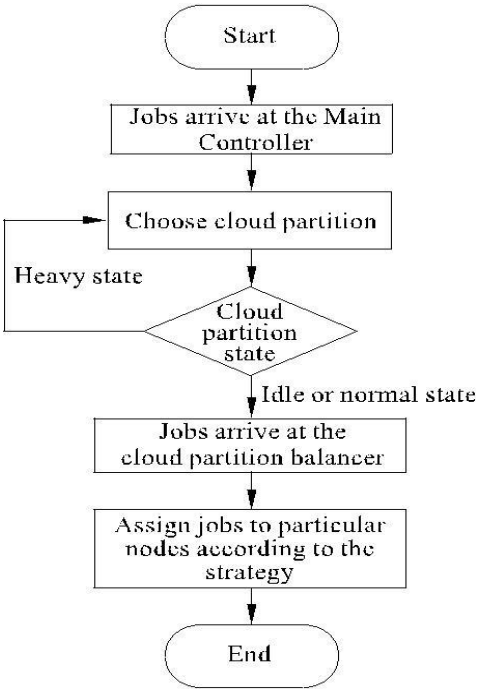

Figure 2. Load Rebalancing Flow chart

VII. LITERATURE SURVEY

Literature survey is the most important step in software development process. Before developing the tool it is necessary to determine the time factor, economy n company strength. Once these things r satisfied, ten next step is to determine which operating system and language can be used for developing the tool. Once the programmers start building the tool the programmers need lot of external support. This support can be obtained from senior programmers, from book or from websites. Before building the system the above consideration r taken into account for developing the proposed system

The express growth of communication technologies and the Internet in particular has changed the way we live and work. It has the possible to transform a big part of the IT industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased. There have been many companies which have plunged deep in this Such as Amazon EC2 [6], Google AppEngine[4], Microsoft Azure[5], Sales force[3], etc. Projections show great future growth of cloud computing. Although estimates vary wildly, a research firm IDC [1] predicts cloud computing will reach worth $42 billion in 2012.The Google File System (GFS) [7], and/or Hadoop Distributed File System (HDFS) [8] are the most common algorithms deployed in large scale distributed systems such as Face book, Google and Yahoo today.Other examples of works in distribute file system are GPFS [18], Frangipani [20] and InterMezzo [5]. In InterMezzo [12], Randles et al [14].

7.1 Requirements

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified authenticated, how it is processed, and how it is displayed as output. In Physical design, following requirements about the system are decided.

1. Input requirement,
2. Output requirements,
3. Storage requirements,
4. Processing Requirements

7.2 Paper Description

The load balancing model given in this article is aimed at the public cloud which has numerous nodes with distributed computing resources in many different geographic locations. Thus, this model divides the public cloud into several cloud partitions. When the environment is very large and complex, these divisions simplify the load balancing. The cloud has a main controller that chooses the suitable partitions for arriving jobs while the balancer for each cloud partition chooses the best load balancing strategy.

VIII. RESULT & DISCURSION

This paper addresses is to balance the Public cloud as per the user satisfaction speed. This cloud partition Divided into four parts as per the requirements based on the time. The load degree results are input into the Load Status Tables created by the cloud partition balancers. Each balancer has a Load Status Table and refreshes it each fixed period **T**. The table is then used by the balancers to calculate the partition status. Each partition status has a different load balancing solution. When a job arrives at a cloud partition, the balancer assigns the job to the nodes based on its current load strategy. This strategy is changed by the balancers as the cloud partition status changes.

IX. CONCLUSIONS

A novel load balancing algorithm to manage rebalancing issue in expansive scale, dynamic and disseminated record frameworks in cloud has been displayed in this paper. Our proposal strives to balance the heap of nodes and diminish the requested development cost however much as could be expected. Our proposal is similar to the unified calculation in HDFS and can be used in Single node or Multi node group environment. The proposed algorithm works in an appropriated way in which nodes perform their heap balance assignments freely without synchronization or worldwide learning in regards to the framework. In a heap equalization cloud the assets can be decently used and provisioned, augmenting the execution of Guide Lessen based applications. The calculation likewise outflanks the contending circulated regarding burden awkwardness component, and development cost. What's more also the paper presents design of a scalable and efficient distributed file system. The system uses a light weight frontend server to manage sessions and forward requests to many name nodes. Our protocol can be directly implemented in current distributed systems such as cloud computing as an overlay. We are currently working on more detailed experiments. We will also implement our system and test it using the Illinois Cloud Computing Tested [6].

REFERENCES

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. Above the clouds: A berkeley view of cloud computing. *Technical Report: Electrical Engineering and Computer Sciences University of California at Berkeley UCB/EECS-2009-28* (Feb 2009), 1–23.

[2] Ciurana, E. *Developing with Google App Engine*. Apress, Berkely, CA, USA, 2009.

[3] Fouquet, M., Niedermayer, H., and Carle, G. Cloud computing for the masses. In *U-NET '09: Proceedings of the 1st ACM workshop on User-provided networking: challenges and opportunities* (New York, NY, USA, 2009), ACM, pp. 31–36.

[4] T.R.V. Anandharajan, Dr.M.A. Bhagyaveni "Co-operative Scheduled Energy Aware Load-Balancing technique for an Efficient Computational Cloud" IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 2, March 2011.

[5] Cerbelaud, D., Garg, S., and Huylebroeck, J. Opening the clouds: qualitative overview of the state-of-the-art open source vm-based cloud management platforms. In *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware* (New York, NY, USA, 2009), Springer-Verlag New York, Inc., pp. 1–8.

[6] Braam, P., Callahan, M., and Schwan, P. The intermezzo file system. In *Proceedings of the 3rd of the Perl Conference, O¨ı£¡Reilly Open Source Convention*, Citeseer.

[7] Schmuck, F., and Haskin, R. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (2002), USENIX Association, p. 19.

[8] Randles, M., D. Lamb and A. Taleb-Bendiab, "A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing," in Proc. IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Perth, Australia, April 2010.

[9] Robinson, D. *Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster*. Emereo Pty Ltd, London, UK, UK, 2008.

[10] Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. *SIGOPS Oper. Syst. Rev. 37*, 5 (2003).

[11] Borthakur, D. *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.

[12] *Improve Namenode Performance*. http://issues.apache.org/jira/browse/HADOOP-3248

[13] Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and West, M. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS) 6*, 1 (1988), 51–81.

[14] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D. Network file system (NFS) version 4 protocol. *Request for Comments 3530* (2003).

