International Journal of Research (IJR) Vol-1, Issue-11 December 2014 **ISSN 2348-6848**

Load Rebalancing in Public Cloud Using Best Partitioning Technique by dividing the Cloud

[1] B.Manohar Sai , [2] G. Rama Swamy

[1]M.Tech Research Scholar, Department of CSE,
[2] Professor, Department of CSE
Priyadarshini Institute of Technology & Science, Chintalapudi, 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.

Keywords-Distributed file system, Cloud, Chunk server, Rebalance Cloud architecture, resource allocation, distributed name node.

1. 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 [2]. There have been many companies which have plunged deep in this, such as Amazon EC2 [6], Google AppEngine[4], Microsoft Azure[5], Salesforce[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. 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. 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 Facebook, Google and Yahoo today. 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 [9]. 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, failure is the norm, and the chunk servers may be upgraded, replaced and added in the system which leads to load imbalance in the distributed file systems. It means that the file chunks are not distributed equitably between the nodes. Distributed file systems in clouds such as GFS and HDFS, rely on central servers (master for GFS and Name Node for HDFS) to manage the metadata and the load balancing. The master rebalances replicas periodically: data must be moved from a data node/chunkserver V to another one if its free space is below a certain threshold. However, this centralized approach can provoke a bottleneck for those servers as they become unable to manage a large number of file accesses. Consequently, dealing with the load imbalance problem with the central nodes complicate more the situation as it increases their heavy loads. In order to manage large number of chunk servers to work in collaboration, and solve the problem of load balancing in distributed file systems, there are several approaches that have been proposed such as reallocating 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 a fully distributed rebalancing algorithm is proposed to solve the imbalance state of the nodes in the system. This algorithm can be integrated with Hadoop Single Node or Multi Node Cluster. Here we have implemented using Single Node Cluster. In this paper, we introduce a load rebalancing algorithm to solve the load balancing problem among all chunk servers (i.e. node in short) in the distributed file system. We rebalance the load by migrating the chunks to the previous node in the system.

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.

2. RELATED WORK

Another popular file system for networked computers is the Network File System (NFS) [11]. It is a way to share files between machines on a network as if the files were located on the client's local hard drive. One of the disadvantages of NFS is that it tries to make a remote file system appear as a local file system, but it's dangerous to rely on that oversimplification. There are many situations in which the use of NFS (compared to a local file system) is not appropriate or reliable. Andrew File System (AFS) [10] is a distributed networked file system which uses a set of trusted servers to present a homogeneous, location-transparent file name space to all the client workstations. AFS has several benefits over traditional networked file systems, particularly in the areas of security and scalability. It is not uncommon for enterprise AFS cells to exceed twenty five thousand clients. AFS uses Kerberos for authentication, and implements access control lists on directories for users and groups. Each client caches files on the local file system for increased speed on subsequent requests for the same file. AFS may not be convenient for large scale file systems such as the once handled by GFS.

Other examples of works in distribute file system are GPFS [18], Frangipani [20] and InterMezzo [5]. 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. GPFS [13] is IBM's parallel, shared-disk file system for cluster computers. GPFS uses a centralized management scheme which can have scalability issues. In InterMezzo [12], the key design decisions were to exploit local file systems as server storage and as a client cache and make the kernel file system driver a wrapper around local file system. However, they rely on existing protocols such as TCP. Besides these systems do not have a good resource allocation which deals with the dynamic link, storage and processing capacities.

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. Honey Bee Foraging algorithm is derived from behavior

of honey bees for finding and reaping food. In order to check for fluctuation in demand of services, servers are grouped under virtual servers having its own virtual queues calculates a profit or reward on basis of CPU utilization which is corresponds to the quality that the bees show in their waggle dance and advertise on the advert board. Each of the servers takes the role of either a forager or a scout. A server serving a request calculates its profit and compare it with colony profit , if profit as high then the server stays at the current virtual servers and on the other hand if profit was low then the server returns to the forger or scout behavior thus balancing the load with the server.

Randles et al. [14] 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 [14] 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 [15] 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 [15].

3. LOAD REBALANCE PROBLEM

We consider a large-scale distributed file system consisting of a set of chunkserver 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 Cf. 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.

4. 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.

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

2. 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.

3. 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.

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

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

6. 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.

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

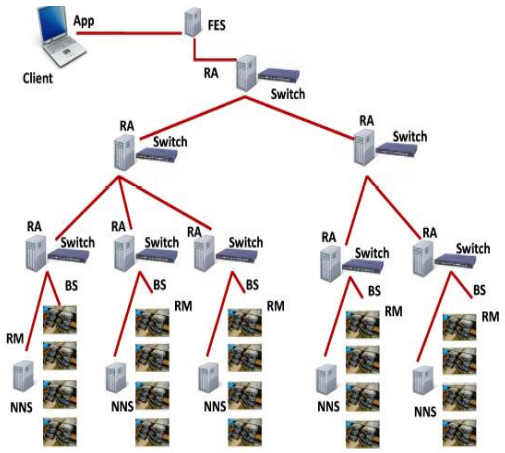

Figure 1: Overview of Our Protocol

5. LOAD REBALANCING ALGORITHM

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

Assumptions

F= {f1, f2 ,f3, ….. fr}

Ni={n1,n2,n3…….ns}

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

```
Input : Set of nodes /chunk servers
Output: Migrating chunks to previous node

for (i=1; 1<=m; i++)
{
if (Ni!=G)
{
for (j=1; j<=s; j++)
{
for (k=1; k<=b: k++) { Ni[ns] = f[ck] }
}
else { return Ni as heavy node} for ( k=1; k<=b; k++)
{
if( ck==NULL) {flag= false; }
else
{ flag=true; } if (ck==flag)
{  Ni+1 migrate its load ck to Ni  }
}
}
}
```

Figure 2. Load Rebalancing Algorithm

6. CONCLUSION

A novel load balancing algorithm to deal with rebalancing problem in large scale, dynamic and distributed file systems in cloud has been presented in this paper. Our proposal strives to balance the load of nodes and reduce the demanded movement cost as much as possible. Our proposal is comparable to the centralized algorithm in HDFS and can be incorporated in Single Node or Multi Node cluster environment. The proposed algorithm

operates in a distributed manner in which nodes perform their load balancing tasks independently without synchronization or global knowledge regarding the system. In a load balance cloud the resources can be well utilized and provisioned, maximizing the performance of Map Reduce based applications. The algorithm also outperforms the competing distributed in terms of load imbalance factor, and movement cost.

REFERENCES

[1] IDC Cloud Computing. http://blogs.idc.com/ie/?p=224.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

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

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

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

[10] 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.

[11] 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).

[12] 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.

[13] 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.

[14] 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.

[15] 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.