[Deshmukh, 3(7): July 2016]
DOI-10.5281/zenodo.58584

ISSN 2348 – 8034
Impact Factor- 4.022

GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

SCALABLE LOAD REBALANCING FOR DISTRIBUTED FILE SYSTEM IN CLOUDS

Mr. Mohan S. Deshmukh[*1] and Prof. S.A.Itkar[2]

[*1]Department of Computer Engineering. Pune University, ME II Year P.E.S's Modern College of Engineering
[2]Department of Computer Engineering, Pune University, Assistant Professor P.E.S's Modern College of Engineering

ABSTRACT

Cloud computing is an upcoming era in software industry. It's a very vast and developing technology. Distributed file systems play an important role in cloud computing applications based on map reduce techniques. While making use of distributed file systems for cloud computing, nodes serves computing and storage functions at the same time. Given file is divided into small parts to use map reduce algorithms in parallel. But the problem lies here since in cloud computing nodes may be added, deleted or modified any time and also operations on files may be done dynamically. This causes the unequal load distribution of load among the nodes which leads to load imbalance problem in distributed file system. Newly developed distributed file system mostly depends upon central node for load distribution but this method is not helpful in large-scale and where chances of failure are more. Use of central node for load distribution creates a problem of single point dependency and chances of performance of bottleneck are more. As well as issues like movement cost and network traffic caused due to migration of nodes and file chunks need to be resolved. So we are proposing algorithm which will overcome all these problems and helps to achieve uniform load distribution efficiently.

Keywords: *Clouds, Distributed File system, Load balance, Movement cost, Network traffic.*

1. INTRODUCTION

Cloud computing is a computing paradigm, where a large pool of systems are connected in private or public networks, to provide dynamically scalable infrastructure for application, data and file storage. With the advent of this technology, the cost of computation, application hosting, content storage and delivery is reduced significantly.

Cloud computing is a practical approach to experience direct cost benefits and it has the potential to transform a data center from a capital-intensive set up to a variable priced environment.

The idea of cloud computing is based on a very fundamental principal of „reusability of IT capabilities'. The difference that cloud computing brings compared to traditional concepts of "grid computing", "distributed computing", "utility computing", or "autonomic computing" is to broaden horizons across organizational boundaries.

1.1 Distributed File System

A Distributed File System (DFS) is simply a classical model of a file system distributed across multiple machines. The purpose is to promote sharing of dispersed files. The resources on a particular machine are local to itself. Resources on other machines are remote. A file system provides a service for clients. The server interface is the normal set of file operations: create, read, etc. on files.

Distributed file systems are key building blocks for cloud computing applications based on the MapReduce 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

MapReduce 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., chunkservers). Each storage node (or node for short) then calculates the frequency of each unique word by scanning and parsing its local file chunks.

1.1.1 LOAD BALACNING IN DFS

In Distributed file systems, nodes simultaneously serve computing and storage functions; a file is partitioned into a number of chunks allocated in distinct nodes so that MapReduce tasks can be performed in parallel over the nodes. However, in a distributed 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 results in load imbalance in a distributed file system; that is, the file chunks are not distributed as uniformly as possible among the nodes. Load balance is critical problem in distributed file system.

1.1.2 LOAD BALANCING APPROACH

Load balancing approaches can be broadly classified as centralized approach and distributed approach.

1.1.2.1 Centralized approach

In centralized option system is dependent on central nodes to manage metadata information of file systems and to balance the loads of storage nodes based on metadata. Centralized approach is based on simplified design and its implementation is also so simple. But as the size of the file increases or overall load of the systems increases centralized approach may face performance bottleneck problem. As its difficult to handle heavy load of access of files. Some of the example of centralized approach is google file system and HDFS federation. In HDFS multiple namenodes are used and file system is manually partitioned. But here also uniform load distribution cannot be achieved.

1.1.2.2 Distributed approach

In this approach unlike central node to balance the load number of nodes are setup for load balancing this approach works successfully for large data files and can achieve as uniform load distribution as possible.

1.1.2.3 Static Load Balancing

In static algorithm the processes are assigned to the processors at the compile time according to the performance of the nodes. Once the processes are assigned, no change or reassignment is possible at the run time. Number of jobs in each node is fixed in static load balancing algorithm. Static algorithms do not collect any information about the nodes. The assignment of jobs is done to the processing nodes on the basis of the following factors: incoming time, extent of resource needed, mean execution time and inter-process communications. Since these factors should be measured before the assignment, this is why static load balance is also called probabilistic algorithm. As there is no migration of job at the runtime no overhead occurs or a little over head may occur. In static load balancing it is observed that as the number of tasks is more than the processors, better will be the load balancing.

1.1.2.4 Dynamic Load Balancing

During the static load balancing too much information about the system and jobs must be known before the execution. This information may not be available in advance. A thorough study on the system state and the jobs quite tedious approach in advance. So, dynamic load balancing algorithm came into existence. The

[Deshmukh, 3(7): July 2016] ISSN 2348 – 8034
DOI-10.5281/zenodo.58584 Impact Factor- 4.022

assignment of jobs is done at the runtime. In DLB jobs are reassigned at the runtime depending upon the situation that is the load will be transferred from heavily loaded nodes to the lightly loaded nodes. In this case communication over heads occur and becomes more when number of processors increase. Dynamic load balancing no decision is taken until the process gets execution. This strategy collects the information about the system state and about the job information. As more information is collected by an algorithm in a short time, potentially the algorithm can make better decision. Dynamic load balancing is mostly considered in heterogeneous system because it consists of nodes with different speeds, different communication link speeds, different memory sizes, and variable external loads due to the multiple. The numbers of load balancing strategies have been developed and classified so far for getting the high performance of a system.

2. COMPARISON BETWEEN SLB & DLB ALGORITHM

Some qualitative parameters for comparative study have been listed below.

2.1 Nature

Whether the applied algorithm is static or dynamic is determined by this factor.

2.2 Overhead Involved

In static load balancing algorithm redistribution of tasks are not possible and there is no overhead involved at runtime. But a little overhead may occur due to the inter process communications. In case of dynamic load balancing algorithm redistribution of tasks are done at the run time so considerable over heads may involve. Hence it clear that SLB involves a less amount of overheads as compared to DLB.

2.3 Utilization of Resource

Though the response time is minimum in case of SLB, it has poor resource utilization capability because it is impractical to get all the submitted jobs to the corresponding processors will completed at the same time that means there is a great chance that some would be idle after completing their assigned jobs and some will remain busy due to the absence of reassignment policy. In case of dynamic algorithm since there is reassignment policy exist at run time, it is possible to complete all the jobs approximately at the same time. So, better resource utilization occurs in DLB.

2.4 Thrashing or Process Dumping

A processor is called in thrashing if it is spending more time in migration of jobs than executing any useful work [1]. As the degree of migration is less, processor thrashing will be less. So SLB is out of thrashing but DLB incurs considerable thrashing due to the process migration during run time.

2.5 State Woggling

It corresponds to the frequent change of the status by the processors between low and high. It is a performance degrading factor [1].

3. Problem Definition

We are interested in studying the load rebalancing problem in distributed file systems specialized for large-scale, dynamic and data-intensive clouds. The terms “rebalance” and “balance” is interchangeable. Such a large-scale cloud has hundreds or thousands of nodes (and may reach tens of thousands in the future). 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

ISSN 2348 – 8034
Impact Factor- 4.022

to normal applications.

3.1 System Architecture

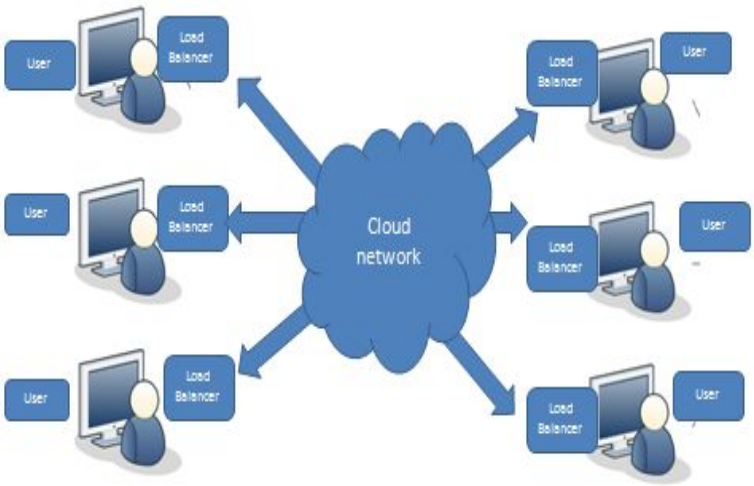

Figure : System Architecture

3.2 Algorithmic Strategy

Input: File (metadata information)

Processing

1. Setup cloud
2. Creation of configuration file.
3. Creation of replication configuration file.
4. web-base application generation
5. File upload/download access
6. File splitting into equal sized chunks

Output: Uniform load distribution among all nodes

3.3 Pseudo Code

Algorithm 1: SEEK (V, ΔL, ΔU):a light node i seeks an overloaded node j

Input : vector V = {S samples}, ΔL and ΔU

[Deshmukh, 3(7): July 2016]
DOI-10.5281/zenodo.58584
ISSN 2348 – 8034
Impact Factor- 4.022

Output : an overloaded node, j

1. Ai ← an estimate for A based on { Aj : j ∈ V};
2. if (Li < (1 − ΔL) Ai)then
3. V ← V ∪ {i};
4. sort V according to Lj (∀j ∈ V) in ascending order;
5. k ← i's position in the ordered set V;
6. find a smallest subset P ⊂ V such that
   (i) Lj > (1 + ΔU) Aj , ∀j ∈ P, and
   (ii) Σ j∈P (Lj − Aj) ≥ kAi;
7. j ← the least loaded node in P;
8. return j;

Algorithm 2: MIGRATE (i, j):

a light node i requests chunks from an overloaded node j

Input: A light node i and an overloaded node j

1. if (Lj > (1 + ΔU) Aj) and j is willing to share its load with i then
2. i migrates its locally hosted chunks to i + 1;
3. i leaves the system;
4. i rejoins the system as j's successor by having
   i ← j + 1;
5. t ← Ai;
6. if (t > (Lj − (1 + ΔU) Ai)) then
   t ← (Lj − (1 + ΔU) Ai);
7. i allocates t chunks with consecutive IDs from j;
8. j removes the chunks allocated to i and renames its ID
   in response to the remaining chunks it manages;

Algorithm 3: MigrateLocalityAware (i, V):

A light node i joins as a successor of a heavy node j that is physically closest to i.

input : A light node i and V = {V1, V2, ▪ ▪ ▪ , VnV }

1. C ← ∅ ;
2. for k = 1 to nV do
3. C ← C ∪ SEEK(Vk);
4. j ← the node in C physically closest to i;
5. MIGRATE(i, j);

ISSN 2348 – 8034
Impact Factor- 4.022

4. EVALAUTION PARAMETERS

In this chapter, we evaluate the performance of the proposed system in Load balancing of distributed file systems in clouds

4.1 Setup

Following parameters given in table 9.1 are considered for the configuration of system. Numbers of nodes are kept as 10 and they are placed in the network. Various different file sizes are used for uploading over the network. Files gets divided into equal sized chunks and uniformly distributed over the all available nodes in the network.

Parameters	Value
Number of Nodes	10
Network switch	1
File Size	100 mb -1000mb
Distribution Time	100,200,300 Milliseconds

5. IMPLEMENTATIONS & RESULT ANALYSIS

When evaluated, Overall time taken to upload the given file over the network is calculated in milliseconds, we compare the time taken with other approaches like centralized approach and distributed approach. It is observed that time taken for efficient load rebalancing approach is more than the centralized approach nut it is definitely less than the distributed approach previously proposed.

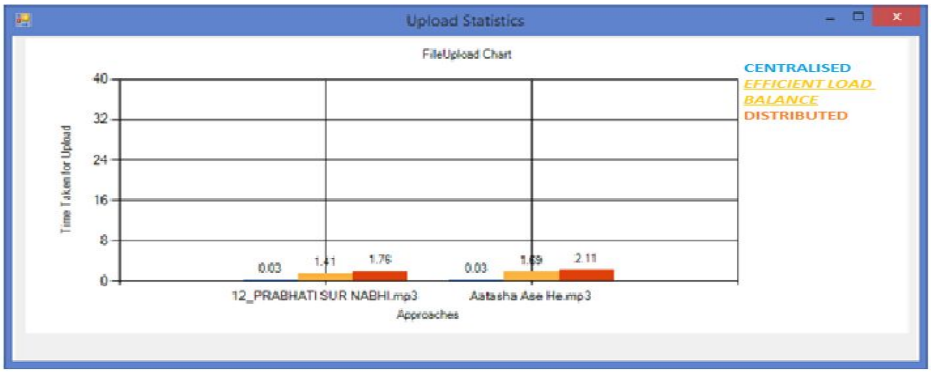

Figure: File Upload

ISSN 2348 – 8034
Impact Factor- 4.022

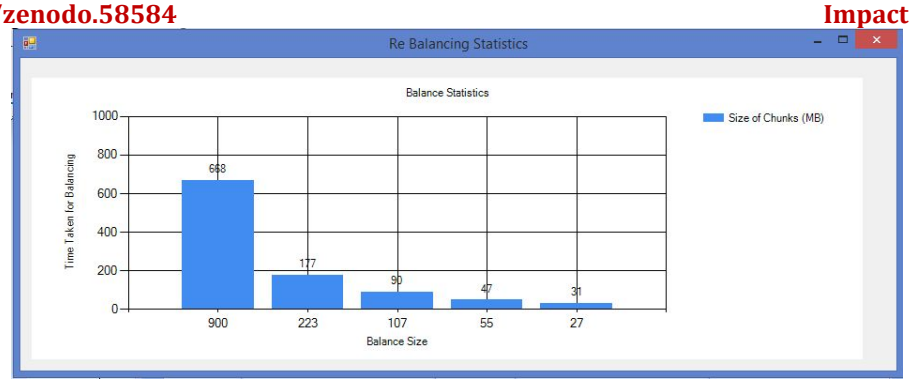

Figure: Rejoining Cost

6. CONCLUSION

In, this Load rebalancing of distributed file system technique, we focused on uniform load distribution of files chunks among all the nodes. We introduced the load rebalancing model, in which it uses less computational on the client side as well as server side. This scheme also minimized the client server communication and leading to less bandwidth consumption. They incur low overhead at the server and requires small and constant amount of communication per load balancing. This scheme verifies the data integrity of file. Our experimental result shows that, such scheme also imposes a significant I/O and computational burden on server. We compare the results of proposed system with the various different approaches already proposed for load balancing; it is observed that our system works very well as compared to other approaches for load balancing.

Future Scope

One problem that is left with load rebalancing of distributed file systems in clouds is node heterogeneity is not achieved in this approaches which limits the system working to same platform nodes only. And if the file size is increased the message overhead is more which should be minimize as much as possible.

REFERENCES

[1] Hung-Chang Hsiao "Load Rebalancing for Distributed File Systems in Clouds " IEEE Computer Society, 2011, p.1–4.

[2] 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.

[3]S. Ghemawat, H. Gobioff and S.-T. Leung "The Google File System," Proc. 19th ACM Symp. Operating systems Principles (SOSP '03), pp. 29-43, Oct. 2003.

[4]Hadoop Distributed File System, http://hadoop.apache.org/ hdfs/, 2012.

[5]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

[6]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

ISSN 2348 – 8034
Impact Factor- 4.022

[7] J.W. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '03), pp. 80-87, Feb. 2003

[8] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou, "Ensuring Data Storage Security in Cloud computing", 17th International workshop on Quality of Service,2009, IWQoS, Charleston, SC, USA, July 13- 15, 2009, ISBN: 978-1-4244-3875-4, pp.1-9G.S. Manku, "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.