ISSN No:-2456-2165

Improving Efficiency of Distributed File System using Map-Reduce Model with Rebalancing Model in Cloud Platform

B.Sabarish, S.Suriya, Suyash Shukla, T.pirithivi Raj, M.Narendran
B.Tech (IV) Year Student , Department of Computer Science and Engineering
SRM Institute of Science and Technology, Chennai, India
Assistant Professor, Department of Computer Science and Technology
SRM Institute of Science and Technology, Chennai, India

Abstract:- A Distributed file system or commonly known as dfs is said to be a major part for setting up and making cloud computing applications. In such file systems such as dfs, nodes continuously working on computing functions and storage facilities and functions; then a file is Partitioned (divided into equal number of parts of equal size)into a number of chunks allocated in different nodes so that Map Reduce algorithm tasks can be applied over the nodes in parallel. However, in a environment such as a cloud computing environment, failure is very evident and often avoided, and nodes in such environment can be enhanced, replaced, or can be added to the current existing system.

Other functions that can be done with files is that they can be created, deleted, and appended dynamically. This leads towards uneven distribution of load that is imbalance problem in a distributed file system; that is, the file parts(Chunks) are not divided between the nodes as uniformly as it should be divided in an ideal state . The present distributed file systems strongly depends upon a central node for reallocating the chunk parts to different nodes. This type of dependence is not considered as good in a large-scale and failure-prone environment as this here because the central load balancer is put under large amount of workload that is linearly scaled with the size of the system, and may cause the performance bottleneck and might become single point of failure. In this paper, technique is presented for improving efficiency of distributed file system using map-reduce model with rebalancing model in cloud platform is presented to deal with the load imbalance problem.

Our algorithm when analyzed with the current approach in production systems and a solution is presented in the literature. The results show and illustrate that our algorithm is comparable with the current existing centralized load rebalancing algorithm and is considerably better than the previous distributed algorithms in factors of load imbalance, movement cost.

I. INTRODUCTION

Clients, data center and distributed servers this are the elements comes under cloud; but there might be some of the issues like fault tolerance, high availability and many more, in between these issues is the establishment of an effective load balancing algorithm. The load can be considered as a load of CPU/network, capacity of the memory. Balancing the node load is to simply distributing the load among sub servers to improve utilization of the resources and response time of the assigned job and to minimize the chances of load imbalance issues. The load balancing technique was mentioned within the existing system. Maintaining and process such a large amount of jobs within the cloud environment is a very tedious task whereas cloud computing are often thought of as a scalable and much more efficient technology therefore it receiving much attention for researchers. Capacities of each node is totally different than alternative every node and also the patterns of job arrival is additionally not predictable because of this load balancing problem work control is incredibly crucial to enhance the performance of the system and to take care of the stability of system. Load balancing algorithms looking on whether or not the system dynamics can be either static or dynamic. Static algorithms don'tuse the system information and are less complex and simple on the other hand dynamic algorithms can prove costly for the system however modification will be done because the system status changes. A dynamic algorithm is used here because of it's flexibility. The load balancing is done on centralized node so that load can be minimized, with the help of load balancing algorithm the weighted nodes load is balanced because of this movement cost is cut down to a minimum. It leads to load asymmetry problem in distributed filing system. To overcome this load inequality problem, distributed Load rebalancing algorithm and map-reduce technique has been applied. Load rebalancing has the more efficiency that makes system more efficient. For load imbalance we use a secure map reduce load rebalancing algorithm that merges with the MD5 with DES Encryption algorithm.

II. LITERATURE SURVEY

A. Iqbalance: https://code.google.com/p/irqbalance

The current present solutions forbalancing load in DHTs leads to a high loss in form of routing state and in form of load movement created by nodes joining the system or leaving the system. In this paper, we present a techniques and use them to introduce set of rules based on Chord, called X0, that can provide load balancing of nodes with minimal incurred loss taking a general assumption that the load is evenly distributed in the memory.

We try to prove that X0 can reach find near to optimal solution for our given problem, while moving little load to maintain the balance and increase the size of the

ISSN No:-2456-2165

routing tables by atmost a constant factor. By performing practical's based upon real-world and user defined scenario, we show that X0 lessens the load imbalance of Chord from O(log n) to a less than 3.6s along with maintaining the number of contact that a node needs to have. we study that the consequences of heterogeneity is reflected, average route length is also reduced as node capacities become more and more heterogeneous. For a real-world of case node capacities, the route length in X0 is asymptotically less than half the path length that as seen in the case of a homogeneous system.

B. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications

A very basic problem that is encountered in a peer-to-peer application is that of the properand correct position of the nodes that stores user data . This paper brings up the concept of *Chord*, a distributed search technique that resolves the aboveproblem. Chord gives solution for just one problem and the solution is: given a key, it maps the key onto a node.

Data location can be used along with Chord by allocating a key with each data part, and storing the key-data pair in the node to which the key points in the memory. Chord adjusts as nodes joins or leaves the system, and can return data foruser-queries given that the system is continuously changing. Results from a written point of view and experiments show that Chord is scalable i.e as wanted by user and as much wanted by user : communication cost and the state of the system is maintained by scaling each node logarithmically with the number of Chord nodes.

C. Scaling in the linux networking stack. Linux Kernel Document.

The Map Reduce model is a programming model and an method of implementation for processing and producing huge amount of data. Users tell a *map* function to manipulate the data and generate a key/data pair to generate a set of in-between key/data pairs, and a *reduce* function that joins all in-between values that have relationship with the same in-between key. Many real world tasks can be expressed and represented using this above model.

Programs written in this way of style are parallelized and run on a large cluster of user systems automatically. The system performing the processing and data manipulation makes note of the splitting up of the input data, performing the program's running across a number of systems, dealing with system failures problem, and managing the as given inter-system communication. This gives an upper-hand to programmers without any prior knowledge in parallel and distributed systems to easily use the resources of a large distributed system.

Our implementation of Map Reduce algorithm works upon on a large cluster of systems and is scalable(as required by user or organization),a general Map Reduce data manipulation deals with and manipulates many terabytes of data on many systems. Programmers find this system as easy to use and work on, as many Map Reduce programs have been already running and more over immense number of Map Reduce jobs are executed on Google's clusters every day.

- *System architecture*

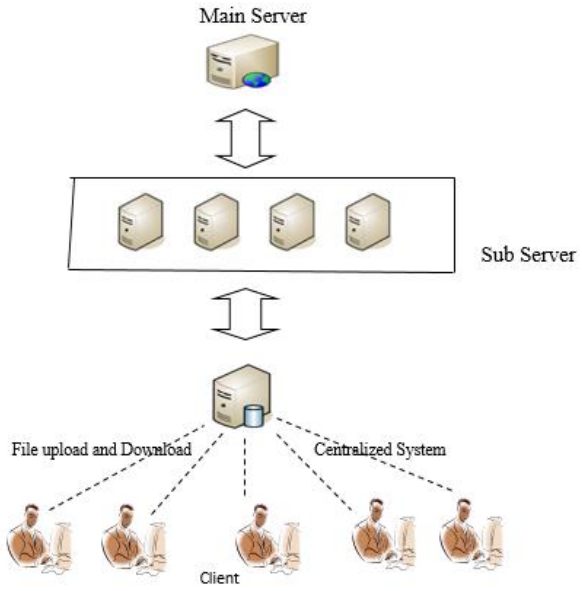

Fig 1:- Overview

The load of each *server* is stable over time, when load balancing process is performed.

D. Hardware Requirments

- Processor : >2GHz
- RAM : 512 MB
- Hard Disk : 80 GB

E. Software Requirements

- Operating System : Windows 10
- Language Used : Java 7
- Tools : NetBeans IDE, MYSQL Server

III. MODULES

A. Chunk Creation

A file is divided into a number of pieces known as chunks situated in different nodes so that Map Reduce operations and functions can be performed upon them simultaneously over the nodes. The load upon a node depends upon the number of chunks the node has kept with itself because the files in a cloud can be created, deleted, and appended by any anonymous user and nodes can be enhanced, changed and increased in the current environment, the file parts may not be divided as uniformly as expected among the nodes .Here our objective is to distribute the chunks as uniformly as can be done among the nodes such that no node has extra or less chunks.

B. DHT Formulation

The nodes performing storage operations are kept near one another as a network based upon *distributed hash tables* (*DHTs*), like, finding a file part can simply imply to quick key search in DHTs, provided that a unique name (or *identifier*) is given to each file part. DHTs gives capability to

nodes to self-organize and self-repair along with offering search facility in nodes without a stop, simplifying the administration and management. The chunk-servers(servers that stores file parts i.e chunks) in our rebalancing method are kept together as a DHT network. DHTs provides a assurance that if a node leaves the environment, then its locally contained chunks are moved to the node which replaces it; if a node joins the system, then it gives the chunks IDs of node immediately to the joining node from its successor to deal with.

C. Load Balancing Algorithm

Our proposed algorithm, takes each one of the chunk-server nodes and it first calculates if it is light-loaded or over-loaded without the information about the system state. A node is said to be light-loaded if the number of file-parts it contains is less than the ideal load that a node can take that is threshold value of load. to be precise, each node gets in touch with randomly selected nodes in the system and creates a vector . A vector is the one that contains entries, and each of this entry has the ID, network address and load status of a randomly selected node.

D. Replica Management

Generally in distributed file systems like Google GFS and Hadoop HDFS, a specific number of duplicate of each chunk is kept in different nodes to have better file availability in any mishappening or if any node failures occurs or any node leaves the system. Our current load balancing algorithm does not sees the duplicates as unique. It is rare that two or more duplicates are kept in a similar node because of the unknowing nature of our here presented load rebalancing method. More over, each light-loaded node checks a number of nodes, each node is selected with a probability of 1/n, for sharing their loads (where n is the total number of nodes that provides storage facility).

IV. CONCLUSION

The load balancing is a very crucial work in a Cloud Computing environment to get maximum output from the existing system resources. We discussed load balancing schemes along with the Map-Reduce Model for improving efficiency of distributed file systems.

On one side the existing load balancing Model provide easiest performance monitoring of file system environment, but it is a difficult task to model the heterogeneous nature of clouds. On the other side, dynamic load re-balancing algorithms are complex to perform but are preferred in a heterogeneous system of cloud.

REFERENCES

[1]. Irqbalance: https://code.google.com/p/irqbalance.
[2]. lookbusy–asyntheticload generator:http://www.devin.com/lookbusy/.
[3]. Scaling in the linux networking stack. *Linux Kernel Document*.
[4]. STREAM bechmark: https://www.cs.virginia.edu/stream/.
[5]. Xen's Credit Scheduler:
[6]. http://wiki.xen.org/wiki/credit scheduler.